SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 2. Nº 12
1250 PTAS.

ARGENTINA 9'50 $
CHILE 3000 $

Por desarrolladores
ESCRITA
para desarrolladores

Y además:

- Tratamiento digital de la imagen
- Técnicas de sonido
- X-Window para Unix
- Sistemas expertos
- Programación C++
- Construcción de demos

Estándar VESA para tarjetas SVGA

DE REGALO
El Libro
"WATCOM 386 C/C++:
Guía del usuario"

Compilación del núcleo de Linux

Dentro de WinEditor 2.0

Demomanía: Efecto de túnel de estrellas

TORNEO DE PROGRAMACIÓN DE DEMOS

9 778411 344792 00012

TOWER COMMUNICATIONS, S.R.L.

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

AGOSTO 1995. Número 12
Es una publicación de

TOWER COMMUNICATIONS, S.R.L.

Editor
Antonio M. Ferrer Abelló
Directora Comercial
Carmina Ferrer
Director de Producción
Carlos Peropadre
Publicidad
Magdalena Pedreño Llorente

••••••

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Luis Martín

Colaboradores
Emilio Postigo, Fernando J. Echevarrieta, Juan M. Martín, José María Peco, José Carlos Remiro, Pedro Antón, Luis Crespo, Juan José Samper, José Galán, Jorge R. Regidor, Agustín Guillén, Enrique Coiras, David Aparicio, Fernando de la Villa, Ignacio Cea.

Edición técnica
Emilio Castellano
Maquetación
Fernando García Santamaría
Tratamiento de imagen
Josefa Fernández Martínez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón
Secretaría de Redacción
Consuelo Jiménez
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Duvial
Impresión
G.D.B.
Distribución
MIDESA

La revista SÓLO PROGRAMADORES no tiene porqué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados. El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792

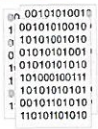
SUMARIO

6 CONCURSO
Sólo Programadores convoca el primer concurso de demos e infografía. Además en éste número se incluye el cupón para participar.

8 NOTICIAS
Lo último y más interesante del mundo informático: compiladores, herramientas, lenguajes, nuevos equipos,... y el comentario de libros de interés.

12 ENTREVISTA
Gaelco, la primera compañía española dedicada a las COIN-OP o "máquinas de bar" nos cuenta su organización, expectativas, y modo de funcionamiento.

15 CURSO DE ENSAMBLADOR
Esta nueva entrega comenta el funcionamiento de las directivas de control completo de segmento, que proporcionaran una nueva herramienta para programar.



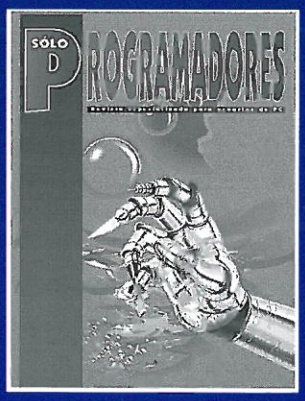
20 ESTÁNDARES GRÁFICOS: NORMA VESA
Que es y como se puede utilizar la norma VESA para tarjetas SVGA, el estándar de las tarjetas gráficas para ordenadores personales.

24 CURSO DE PROGRAMACIÓN BÁSICA
Este capítulo incide sobre la subdivisión de los programas en subprogramas para conseguir que estos sean mucho más estructurados.

29 CURSO DE UNIX: X-WINDOWS
Comienza en este número una serie de artículos sobre el sistema operativa Unix. En esta primera entrega se describe en detalle el entorno X-Windows.

34 TRATAMIENTO DIGITAL DE LA IMAGEN
En esta entrega se estudian las funciones de tratamiento digital de la imagen que van a permitir a un ordenador personal funcionar como una estación.

38 CÓMO HACER UNA DEMO
Esta nueva entrega ilustra la utilización del efecto de túnel de estrellas, un efecto muy vistoso y utilizado, que al mismo tiempo es relativamente sencillo.



SUMARIO

42 **TÉCNICAS DE SONIDO**
Una vez explicados los conceptos previos, es el momento de lanzarse a la interpretación de la partitura de un módulo.

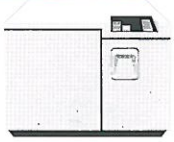
47 **SISTEMAS EXPERTOS**
Uno de los puntos más controvertidos y discutidos en el mundo de los sistemas expertos son las diferentes tendencias sobre para representación del conocimiento.

53 **PROGRAMAS RESIDENTES BAJO D.O.S.**
Este nuevo capítulo analiza varios de los aspectos relacionados con los programas TSR como la gestión de memoria, la utilización de la VGA, etc.

56 **CURSO DE PROGRAMACIÓN BAJO WINDOWS**
La forma más conveniente de introducir datos es mediante las cajas de diálogo. En esta ocasión se explica la utilización de las mismas.

62 **SISTEMA OPERATIVO LINUX**
Este artículo analiza la compilación del núcleo de Linux para poder así aprovechar el Hardware instalado, o realizar modificaciones del ya existente.

68 **GRANDES SISTEMAS**
Una vez redireccionada la salida de un comando a un fichero, hay que ver como tratar dicho fichero para navegar mediante paneles de selección.



72 **CLIPPER**
Tras el nivel alcanzado en los últimos artículos, se hace necesario dejar claros algunos conceptos básicos de este potente lenguaje de programación.

76 **PROGRAMACIÓN EN C++**
Continuando con el sistema de ventanas, este artículo explica los mecanismos para dibujar en una ventana, así como sus problemas y las posibles soluciones.

81 **CORREO DEL LECTOR**
Esta es la sección dedicada a la resolución de las dudas y consultas que le surgen a nuestros lectores en diversos ámbitos de la programación.

NOTA
En el CD-ROM que se regalaba en el nº 10 de Sólo Programadores
faltaban dos ficheros del Linux que se han incluido en el disquete en el directorio CD_LINUX

1 er Torneo de Demos e Infografía

El tan esperado campeonato de demos ya está aquí.

Como ya anunciamos el mes pasado, Sólo Programadores organiza un concurso para los más hábiles y creativos de la programación. Se trata de un campeonato de **Demos** e **Infografía**. Dentro del apartado de demos se incluye también el de **Intros** (como las demos pero más pequeñas). Además del apartado de Infografía se encuentran el de **Vídeo** y el de **Morphing**, de los cuales se habla más adelante.

Existe un suculento premio de **250.000 pesetas en metálico.** para los ganadores además de numerosos paquetes de software.

El torneo abre sus puertas a los intrépidos de la programación desde este mes de agosto. Durante varios números recogeremos los trabajos realizados por todos aquellos que se animen a participar.

BASES DEL CONCURSO

- Para concursar es imprescindible rellenar y enviar la tarjeta de participación que se adjunta en estas páginas.
- Se podrán enviar uno o varios trabajos en cualquiera de las categorías, que son las siguientes:
- El jurado estará compuesto por la redacción de Sólo Programadores.

- DEMOS:

- No entrarán en el concurso aquellas demos que hayan obtenido primeros puestos en campeonatos internacionales. Sin embargo se podrán enviar para incluirlas en el CD-ROM en el que se publiquen los resultados del concurso.
- Existen varias ramificaciones en la categoría de demos:

INTROS

Son demos de longitud mucho más reducida. Habrá dos grupos: de tamaño menor o igual a **4 Kb** (4.096 bytes) y hasta **64 Kb** (65.536 bytes). Tradicionalmente las primeras no suelen incluir sonido, mientras que las segundas suelen aportar alguna melodía o efecto sonoro.

Las longitudes se refieren al tamaño del fichero ejecutable (que vemos desde el DOS). El archivo .EXE o .COM puede estar empaquetado con la utilidad PK-lite. Dicha utilidad comprime un archivo EXE y genera otro EXE de menor tamaño. Este último cuando se ejecuta se descomprime en RAM en tiempo real quedándose con su tamaño original que se supone es superior. Así Intros que ocupan 100 Kb se reducen a 60 Kb con la utilidad PK-Lite o similar, y pueden entrar en la categoría de 64 Kb.

DEMOS

Se entiende por longitud de la demo la suma de las longitudes de los ficheros necesarios para la ejecución de la demo, bien sea un sólo archivo EXE o uno ejecutable más otros de datos. Dichos ficheros también pueden utilizar técnicas de autodescompresión en memoria como las comentadas en el apartado anterior.

Las demos tendrán un tamaño superior a 64 Kb. La única limitación es la de 4 Mb de memoria RAM total para ejecutar la demo. Se pueden programar demos con extensores de DOS como el DOS4GW de Watcom puesto que el ordenador de prueba será un **486 DX2-66**. Dichos extensores deberán ser enviados en el disquete para poder evaluar la demo.

No se contabilizará como válida la longitud del archivo si está comprimida con uno de los compresores habituales (ARJ, LZH, ZIP, etc.).

- Se enviarán dos copias de la demo. Esto se puede hacer de varias formas. Una consiste en copiar el archivo de la demo dos veces en el disquete (con distinto nombre). Otra forma es enviar dos discos con una copia del archivo en cada uno.

Si es necesario la demo estará comprimida, y si supera la cantidad de 1,4 Mb se podrá empaquetar por volúmenes. **Sólo se admitirán las siguientes compresiones: ARJ, ZIP y LZH.**

En los disquetes irá una pegatina como la que aparece en la **figura 2**. En primer lugar irá el "handle" o nombre del participante/grupo para el concurso y en las casillas se pondrá una X indicando las categorías a las que pertenecen los trabajos enviados en dichos disquetes. Por ejemplo, si un grupo de amigos envía una Intro de 4 Kb en un disquete, pondrá en la pegatina el nombre del grupo de programación y una X en DEMOS, porque la categoría INTROS pertenece a DEMOS.

- INFOGRAFIA:

En las dos categorías, **Imagen** y **Animación**, se admitirán solamente trabajos realizados con programas de infografía o imagen sintética (3D Studio, POV, Topas, etc.).

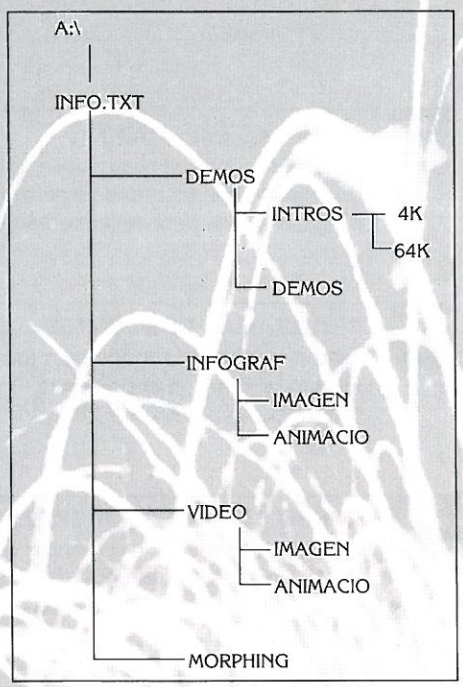


Figura 1.

- **Imagen.** Aquí habrá sub-categorías para englobar los trabajos en función de la resolución y número de colores.
  Sólo serán válidos los trabajos en formato TIF, GIF, JPG y TGA.
  Además de incluir la infografía final se adjuntará obligatoriamente al menos una pantalla del trabajo en fase de construcción. Esto se hará para certificar que el trabajo está siendo realizado por su correspondiente autor.
- **Animación.** En el apartado de infografía en movimiento los trabajos serán admitidos en cualquier formato siempre y cuando se incluya el programa (DOS o Windows solamente) que visualize la animación en un PC 486-DX con 8 Mb RAM y tarjeta True Color VESA.

- VIDEO:

Aquí hay dos grupos similares a la categoría anterior: **Imagen** y **Animación**.

- **Imagen.** Es para los trabajos o composiciones fotográficas en los que se hallan empleado programas específicos para el fotomontaje (Photoshop, Picture Publisher, Photostyler, etc.). Se admiti-

(HANDLE O NOMBRE DE PARTICIPACIÓN)

DEMOS [ ] INFOGRAFIA [ ]

VIDEO [ ] MORPHING [ ]

Figura 2.

rán sólamente los formatos TIF, GIF, JPG y TGA.

- **Animación.** Aquí se engloban los trabajos que hayan sido digitalizados como vídeos caseros de humor o imágenes con cierto tratamiento digital (efectos de cortina, fades, etc.). Se admitirán solamente trabajos en formato AVI (Video for Windows) y MOV (QuickTime).

- MORPHING:

Se entiende por trabajos de morphing aquellos en los que a través de una secuencia de vídeo se observan transformaciones de objetos, animales o personas. Para estas secuencias no se permite utilizar personajes del ámbito público (políticos, actores, etc.). Sólo se admitirán ficheros AVI, MOV, FLI, FLC y programas que hagan Morphing en tiempo real siempre y cuando hayan sido programados con rutinas propias de los concursantes.

SOBRE EL MATERIAL ENVIADO

Sólo se admitirá el soporte de disquete y cintas de streamer compatibles QIC-80 para enviar los trabajos. Si se desea que se devuelvan los disquetes y las cintas de streamer hay que adjuntar los correspondientes sobres ya franqueados para que Sólo Programadores realice la devolución.

Los disquetes o las cintas donde se envíen los trabajos tendrán una estructura de directorios como la que se muestra en la **figura 1**.

En cada subdirectorio se colocarán el/los programa/s que vayan a participar. En el fichero INFO.TXT se escribirá en formato ASCII el nombre y teléfono de contacto del representante, así como una breve descripción de cada uno de los trabajos que se envíen.

LOS PREMIOS

Se le darán **250.000 pesetas en metálico** a la mejor y más espectacular **Demo o Infografía**. Los finalistas de las otras categorías recibirán numerosos programas de regalo.

TARJETA DE PARTICIPACIÓN

Nombre:.....Apellidos:.....Edad: .....

Domicilio: .....Cód.postal:.....Teléfono:.....

Pseudónimo o handle para el concurso: .....

CATEGORÍA DE PARTICIPACIÓN

↓ (En cada modalidad se pondrá el número y los nombres de los trabajos enviados) ↓

- Demos [ ]: _____
- Infografía (ficheros .GIF, .TGA, .TIF, FLI, FLC, etc.)
  - Imagen fija [ ]:_____
  - Animaciones [ ]: _____
- Vídeo digital (ficheros .GIF, .TGA, .TIF, .AVI, .MOV, etc.)
  - Imagen fija [ ]: _____
  - Animaciones [ ]: _____
- Morphing [ ]: _____

NORMATIVA PARA EL CONCURSO

- Los trabajos enviados podrán estar realizados por una o varias personas. Si se trata de varias personas solamente el representante aparecerá en la tarjeta de participación. Los datos personales de los otros componentes del grupo serán adjuntados en un folio aparte escritos a máquina.
- No es necesario incluir los fuentes de los programas enviados.
- La responsabilidad de los trabajos enviados atañe tanto al representante que aparece en la tarjeta de participación como a los otros componentes del grupo si es que los hay. Ambos tendrían que responder ante la ley si terceras personas reclamasen la autoría del material o trabajos enviados para este concurso.
- La propiedad intelectual y los derechos de explotación del material enviado pertenecen tanto al representante del grupo como a los demás componentes si los hay. Con la participación en este concurso se concede a la editorial Tower Communications el permiso para publicar en CD-ROM el software enviado para dicho concurso.
- La redacción de Sólo Programadores se reserva el derecho de descalificar a algún participante por un mal funcionamiento del programa enviado o porque dicho programa contenga algún virus informático. También quedarán descalificados aquellos trabajos cuyo contenido sea de mal gusto.
- Enviar esta tarjeta de participación implica aceptar las bases del concurso y la normativa que se estipula en estos apartados.
- La participación en el concurso no será válida si falta algún dato, si hay información incorrecta o bien si la tarjeta de participación es una fotocopia y no la original que viene en la revista. Tampoco será válida la participación en el concurso si la tarjeta enviada no está firmada por el representante.
- La revista Sólo Programadores se reserva el derecho de modificar o ampliar las bases del concurso sobre la marcha si fuese necesario.

Acepto, en representación propia o del grupo, las bases del concurso que aparecen en el número 12 de Sólo Programadores.

Firma del representante:

KIT DE INTEL PARA DESARROLLADORES DE NATIVE AUDIO

Intel Corporation anuncia un kit de desarrolladores para Native Audio, la tecnología de proceso de señales Nativo desarrollada por Intel en colaboración con otras empresas del sector. Dicha tecnolog'a proporciona funciones de sonido avanzadas sobre máquinas PC basadas en el procesador Pentium, que operan en entorno Windows. Entre las características más importantes que ofrece Native Audio figuran:

- Soporte de compresión y descompresión ADPCM y TrueSpeech-1.
- Mezcla digital de múltiples fuentes de audio.
- Arquitectura extensible para soporte de síntesis de instrumentos MIDI por software, audio 3D, efectos especiales y algoritmos avanzados de compresión.
- Soporte para grabación y reproducción simultánea.
- Compatibilidad total con Windows 3.1 y Windows 95.

El kit de desarrollo para Native Audio ofrece un conjunto de herramientas entre las que se incluyen minifiltros de compresión ADPCM y TrueSpeech, códigos fuente de muestra, programas de verificación y la tarjeta de sonido full-duplex de 16 bits CS4231 codec.

La unidad de Native Audio también está incluida en el kit de desarrolladores IA-SPOX de Spectron Microsystems.

Para más información:
Intel Corporation Iberia
Paseo de la Castellana, 39 - 1º
28046 Madrid
Tlf: (91) 308 25 52
Fax: (91) 310 54 60
E-mail: nsp@intel.com

BAY NETWORKS ADQUIERE CENTILLION NETWORKS

Bay Networks ha anunciado la firma de un acuerdo de compra de Centillion Networks, suministrador líder de productos de conmutación Token Ring y Token Ring-a-ATM. Como consecuencia de esta adquisición, Centillion Networks pasará a ser una unidad operativa independiente de Bay Networks, centrando su actividad en la conmutación Token Ring y los mercados Token Ring-a-ATM. Esta adquisición representa la continuación de un acuerdo firmado en enero de 1995 por ambas compañías.

Por otro lado, Bay Networks también ha anunciado que sus *router* Backbone Node han obtenido el primer lugar en el banco de pruebas de prestaciones de integración SDLC/LAN de Tolly Group.

Para más información:
Bay Networks
Ricardo Miranda-Naón
Tlf: (91) 742 50 13

SOLUCIÓN EMS DE SYBASE PARA LA GESTIÓN DE MENSAJES EN ARQUITECTURAS CLIENTE/SERVIDOR

Sybase ha presentado una estrategia y nuevas capacidades para Sybase Enterprise Messaging Services (EMS), una tecnología creada para la gestión de mensajes en entornos Cliente/Servidor.

La solución EMS en su versión 3.1 incorpora Quality Services, que permite a las aplicaciones solicitar una categoría y calidad en el servicio. Ofrece una gestión de mensajes fiable para todas las redes y está perfeccionado para la informática móvil.

En el transcurso del presente año,

Sybase tiene previsto el lanzamiento de dos versiones mejoradas de EMS: Sybase EMS 3.2 con un mejor rendimiento y agentes de comunicación adicionales, y Sybase EMS 4.0 con soporte para multiproceso simétrico (SMP), APIs adicionales y soporte para más sistemas operativos y redes PDA. Para 1996 se ha previsto la inclusión de la tecnología de gestión de mensajes EMS en los APIs Open Client y Open Server de Sybase.

Contacto de prensa:
Marcos Carmena
Spanish Executive Information Service (S.E.I.S.)
Tlf: (91) 308 43 15
Fax: (91) 410 38 76

NUEVOS DETALLES SOBRE EL PROCESADOR P6

Intel ha hecho públicos nuevos detalles sobre P6, un procesador que podrá ser empleado como unidad central de proceso en la creación de un nuevo tipo de ordenadores multiprocesador. P6 utiliza un bus transaccional de 64 bits que completa el principio de ejecución dinámica anunciado por la compañía en febrero.

El procesador P6 toma la forma de un encapsulado formado por dos chips. El procesador está estrechamente ligado con una memoria caché de segundo nivel (L2), permitiendo un mayor procesamiento interno, y reduciendo al mismo tiempo las solicitudes hacia el exterior.

El bus de tratamiento transaccional es como una pequeña red de comunicaciones distribuida en capas (física, de transporte y de datos), la cual es totalmente invisible para el software.

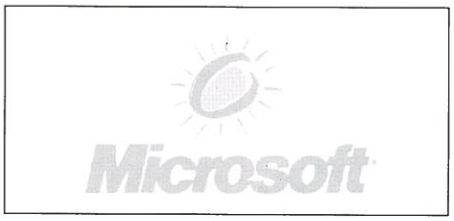
Entre las características del bus del P6 destacan:
- Velocidad de bus de hasta 64 MHz, con un corazón funcionando a 2, 3 ó 4 veces la velocidad del bus.
- Utilización del protocolo MESI
- Bus compartido con solicitudes y respuestas tratadas en *pipeline*
- Control de memoria "snarfing"
- Soporta hasta 8 cargas a 66 Mhz
- No se precisan elementos hardware adicionales para interconectar hasta cuatro P6 en un sólo bus

Para más información:
Intel Corporation Iberia
Pso de la Castellana, 39 - 1º
28046 Madrid
Tlf: (91) 308 25 52
Fax: (91) 310 54 60

MICROSOFT PRESENTA VISUAL C++ VERSIÓN 1.52

Ya está disponible el sistema de desarrollo Visual C++ para Windows versión 1.52 de 16 bits. Este sistema incorpora en las MFC (Microsoft Foundation Classes) el soporte para programación OLE, ODBC y la tecnología de asistentes. La versión 1.52 también está disponible para los miembros del programa de suscripción a Visual C++ que hayan recibido la versión 2.1 en la actualización en CD-ROM.

Algunas características de Visual C++ 1.52:



- Soporte de programación Windows y MS-DOS
- Nuevas MFC de 16 bits: soportan Simple MAPI, Windows Sockets y páginas de propiedad
- Soporte de bases de datos: Access 2.0, Microsoft SQL Server 4.21, FoxPro, Paradox 4.x, dBase III y IV, Excel 5.0 y Btrieve
- Incluye el MFC Migration Kit

Esta versión de Visual C++ requiere un 386 o superior con 8 Mb de RAM que ejecute MS-DOS y Windows 3.1. Así mismo se precisa una unidad de CD-ROM, tarjeta de vídeo VGA o superior y al menos 40 Mb libres en el disco duro.

Para más información:
Microsoft Ibérica
Tlf: (91) 804 00 96

SYBASE PRESENTA NUEVOS PROGRAMAS DE SOLUCIONES PARA LA INDUSTRIA

Sybase Inc. ha presentado el programa Industry Solutions, orientado a las empresas que experimenten un cambio rápido en su negocio. Los primeros sectores incluidos en Industry Solutions son las industrias sanitarias, de telecomunicaciones, petrolíferas, aseguradoras y de servicios financieros.

Los programas de soluciones para la industria de Sybase están formados por expertos de la industria, y dedican recursos a toda la empresa, incluyendo ventas, *marketing*, ingeniería, canales y servicios profesionales. El modelo Industry Solutions está basado en el apoyo a sus socios para que desarrollen sus aplicaciones basadas en Sybase. Los socios de Open Solutions de Sybase ofrecen una amplia selección de proveedores, ISVs e integradores de sistemas, tanto para aplicaciones horizontales como verticales.

Contacto de prensa:
Marcos Carmena
Spanish Executive Information Service (S.E.I.S.)
Tlf: (91) 308 43 15
Fax: (91) 410 38 76

SEMINARIOS BORLAND DELPHI

Durante los meses de junio, julio y septiembre Borland International realiza unos seminarios sobre Borland Delphi, uno de sus productos estrella. Los seminarios están orientados a programadores avanzados y profesionales que deseen conocer en profundidad dicho producto. Están dirigidos por profesionales de la programación formados por los propios creadores de Borland Delphi a través del programa internacional Tech-Transfer de la compañía.

La duración de los seminarios es de dos jornadas completas, en las que se tratan temas como el entorno de programación Delphi, los componentes visuales VCL, conectividad, BDE, componentes QUERY, OLE 2.0, multimedia (MCI) y utilización de DDL y VBX.

La cuota de inscripción es de 95.000 pesetas (IVA no incluido). Los asistentes obtendrán un diploma oficial Borland de asistencia.

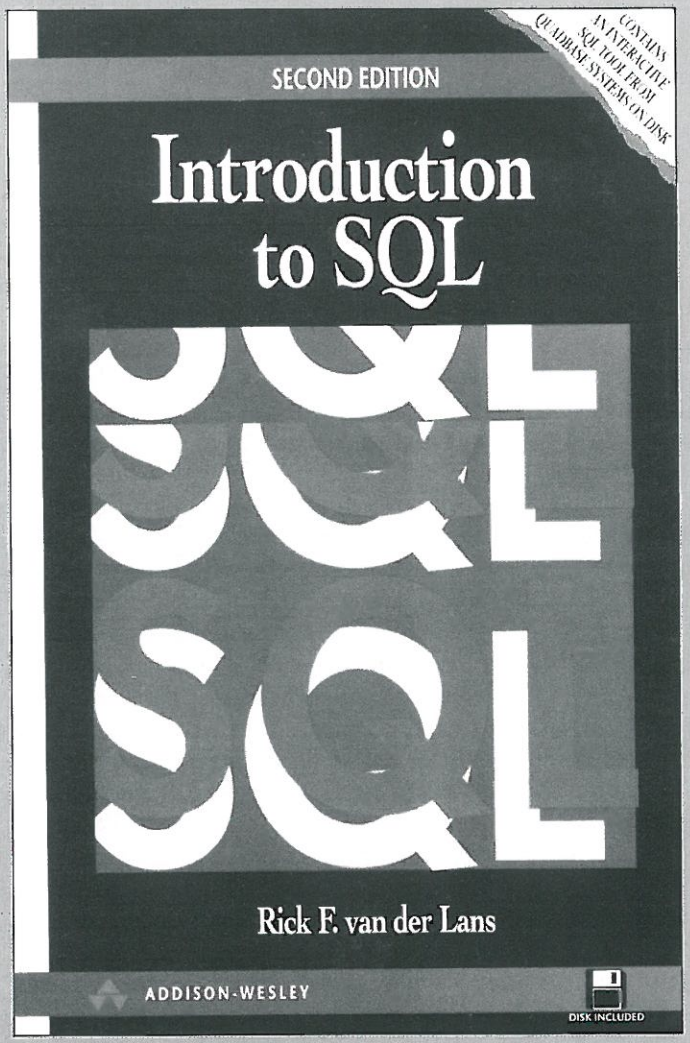
Información y suscripciones:
Tlf: (91) 661 50 10

LIBROS

INTRODUCTION TO SQL (SEGUNDA EDICIÓN)

El lenguaje de consulta estructurado (SQL - Structured Query Language) se ha establecido como un estándar para el manejo de bases de datos relacionales. La obra está diseñada como un libro de texto, dirigida a todo tipo de personas interesadas en SQL con conocimientos generales en programación y bases de datos. En sus páginas se tratan todos y cada uno de los aspectos del lenguaje, con abundantes ejemplos referentes a un posible caso de la vida real: una base datos de un club de tenis. Es un texto eminentemente práctico, que incluye un disco con todos los ejemplos del libro y una herramienta SQL para practicar delante del ordenador.

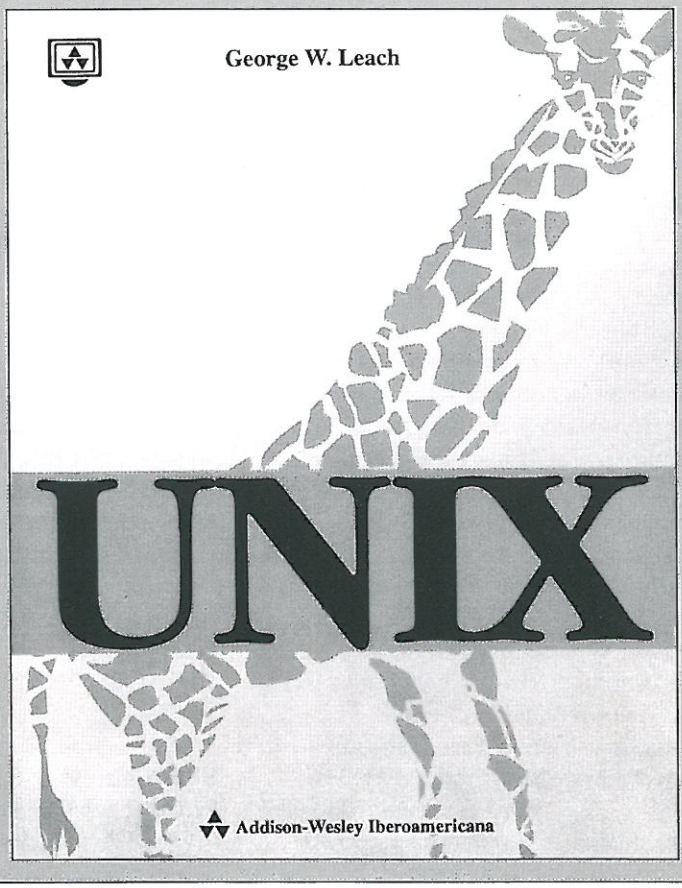
Editorial: Addison-Wesley
Autor: Rick F. van der Lans
491 páginas
Idioma: Inglés
Precio: Consultar



UNIX

Este libro de George W. Leach es una breve introducción práctica al complejo sistema operativo UNIX. A diferencia de otros libros del mismo estilo, el objetivo que se persigue no es la mera descripción del sistema operativo en sí. La obra está enfocada al usuario, de manera que sea capaz de aprender los rudimentos del UNIX sin entrar en aspectos como la filosofía de diseño del sistema. Los primeros seis capítulos del libro proporcionan al lector los conocimientos básicos necesarios para utilizar UNIX en la práctica; el resto del libro está dedicado a ampliar los conocimientos adquiridos en los primeros capítulos. Trae diversos ejercicios y ejemplos, acompañados de la salida por pantalla producida por los mismos para aquellos que no disponen de un sistema UNIX donde practicar.

Editorial: Addison-Wesley Iberoamericana
Autor: George W. Leach
286 páginas
Idioma: Castellano
Precio: Consultar



EL POZO DE LA OCA

Yo envidio a los zapateros. Me explico: lo bueno que tiene ser zapatero es que los zapatos son algo inmutable. Este año se pueden imponer los zapatos de plataforma, o los tonos crudos del esparto, mientras la locura colectiva por las botas tipo Panama Jack experimenta un lento y sereno declive, pero el zapatero o zapatera apenas tendrá que actualizar sus conocimientos. Y esto es así porque la configuración básica del zapato, esto es, su arquitectura, se ha mantenido y se mantendrá inalterada durante bastantes, bastantes años.

Los desarrolladores, en cambio, cada vez que nos metemos en un proyecto estamos haciendo una apuesta. Yo mismo llevo cuatro años usando la herramienta X, y ya estoy empezando a mosquearme porque mi curriculum, así a vista de pájaro, parece que lleva escrito con rotulador permanente "Experto en X", y nada más que "Experto en X". Y lo malo es que yo frecuento con asiduidad las ofertas de trabajo de las páginas salmón del domingo, por aquello de no perder la esperanza en el príncipe azul, y cada vez veo menos anuncios en los que se precisen expertos en X. Creo que ha llegado el momento de comprarme un libro de lo que sea (alguna de las palabras mágicas que están empezando a verse escritas sobre fondo salmón) y autoformarme en casa. Si no, ya me veo víctima de uno de los peores síndromes del programador: el síndrome del pozo de la oca.

Sí, el pozo de la oca es ese sitio donde se queda uno esperando un turno mientras los demás tiran el dado y mueven ficha. Hay herramientas de programación, o plataformas hardware, que tienen esa misma propiedad bloqueante. Por eso hay que estar constantemente alerta, porque el campo de la informática donde trabajamos puede estar convirtiéndose en un pozo de la oca en este preciso momento.

Cuando entré en mi compañía de servicios, mi compañera Nieves formaba parte del grupo de desarrollo de proyectos de un terminal financiero específico, una arquitectura de concentradores de comunicaciones y un procesador con puestos de trabajo conectados a una red propietaria. Un sistema antiguo pero muy probado, que se utilizaba ampliamente en muchas entidades bancarias. Yo vi como el grupo de desarrolladores iba mermando en número a medida que la plataforma se iba haciendo más obsoleta y los clientes iban actualizándose a las ahora omnipresentes redes de "pecés". A medida que esto ocurría, los que quedaban se iban haciendo más y más imprescindibles. Era necesario preservar el mantenimiento que se había garantizado a los pocos y recelosos usuarios de aquel equipo ya jurásico.

Nieves estaba entre esos supervivientes. Se encontraba cómoda: le habían subido el sueldo, llevaba mucho tiempo trabajando con las herramientas de desarrollo específicas del entorno y trabajaba con familiaridad y soltura. A medida que sus compañeros iban desertando, ella quedaba como única experta en un sistema que no le guardaba secretos. Finalmente, el equipo dejó de fabricarse y Nieves quedó encargada sólo del mantenimiento de los sistemas que quedaban. Y ya se sabe lo que pasa con la moral a bordo cuando a uno le ponen a hacer mantenimiento.

Para cuando Nieves quiso escapar de la tela de araña, ya no pudo ser por las buenas. A punto estuvo de abandonar la compañía: fue sólo cuando tuvo en la mano una oferta de trabajo en otra empresa, con menos categoría y sueldo, cuando la tomaron en serio y le ofrecieron una senda gradual de migración a un equipo de desarrollo en pecés, a cambio de no dejar desatendido un puesto que ya sólo ella podía desempeñar y al que le quedaba poco tiempo de vida. Hoy ha conseguido dejar atrás esa etapa, pero su carrera profesional lleva varios años de retraso. Su valía y experiencia han sido desaprovechadas.

Esté atento a los síntomas precoces del síndrome del pozo de la oca. Las ofertas de empleo en su campo ralean, su puesto de trabajo se desempeña con una calma narcótica, su jefe ya no le grita, o peor aún, grita a otros; el mantenimiento supera con mucho al desarrollo. Ha llegado el momento de cambiar de vestuario, aprender nuevos trucos y buscar amor en otro nido. Hay que mantenerse atractivo y frecuentar los lugares de alterne. En informática no existe eso que llaman una relación estable. Estudie, y si cree que después de la carrera ya no tiene ninguna intención de volver a hincar los codos, deje este oficio y hágase zapatero. Tiene más futuro.

ENTREVISTA

GAELCO, LA OPCIÓN EUROPEA

Fotógrafo: David Moreno Aillón

En este número, Sólo Programadores entrevista a Gaelco S.A., la principal empresa europea fabricante de COIN-OP, nombre que reciben las populares máquinas de videojuegos de salón, popularmente conocidas como "máquinas de bar".

Gaelco fue fundada por tres ingenieros electrónicos, que tienen en su haber el nada desdeñable honor de haber sido los creadores de la primera COIN-OP fabricada en Europa. Hace unos 10 años, tras haber trabajado para otras compañías, decidieron independizarse y fundaron Gaelco. Bajo este sello han lanzado máquinas de gran éxito, como "World Rally" o "Thunder Hoop", máquinas que les han proporcionado un gran prestigio tanto dentro como fuera de España. En estos momentos están preparando su desembarco en el mundo de los juegos tridimensionales; la cuestión es no quedarse atrás.

¿Cómo empezasteis en esto?

Gaelco fue fundada por un equipo de ingenieros que trabajó con anterioridad en otra empresa, que fue la pionera en Europa en el campo de las COIN-OP, y toda la experiencia que adquirieron en esta compañía la aplicaron cuando formaron Gaelco en su día.

¿Qué estructura tiene Gaelco?

En Gaelco tenemos una estructura similar a la de cualquier otra empresa del sector. Tenemos dos departamentos, uno de hardware y otro de software.

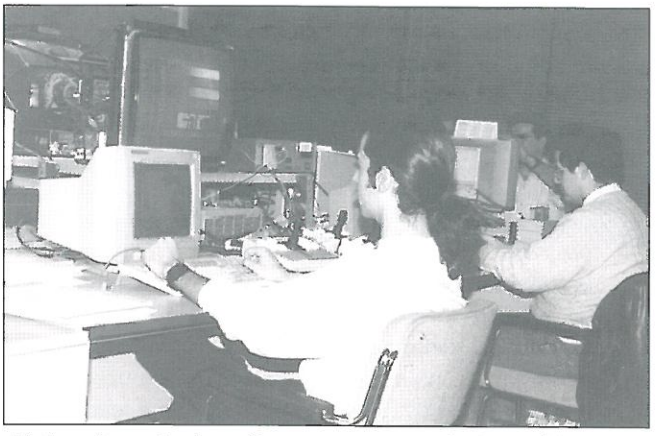
El departamento de hardware se encarga del diseño de las placas de juegos mientras que el departamento de software está subdividido en otros dos departamentos, uno encargado de los gráficos y otro encargado de la programación. Prácticamente todas las empresas del ramo tienen la misma estructura, aunque hay algunas que no tienen departamento de hard; en este caso lo compran a terceros.

En las máquinas de videojuegos, el hardware, ¿Está ya predefinido o se rediseña en función del juego?

Existe un hard básico que es igual para todos los juegos, hard que tiene una vida de dos o tres años; después, en cada juego, este hard sufre algunas modificaciones para adaptarlo a las necesidades de ese juego. Es decir, aunque no diseñamos una placa nueva para cada juego, tampoco hay dos placas iguales. Aparte de esto, cada dos o tres años se produce un cambio de tecnología que obliga a rediseñar completamente el hardware.

El precio de las placas, ¿es fijo o varía según el juego?

Una vez que tienes definido el hardware, la mayoría de los componentes son fijos, por lo que el coste también lo es.



El departamento de software se encarga de los gráficos y la programación.

Sin embargo, existen unos factores que alteran el precio final del producto, principalmente la cantidad de memoria que necesite la placa, que está determinada básicamente por la cantidad de gráficos que tenga el juego. En general, la cantidad de gráficos es lo que más hace oscilar el precio.

El mercado del videojuego, ¿Es muy competitivo? ¿Resulta muy difícil introducirse en el mercado japonés o americano?

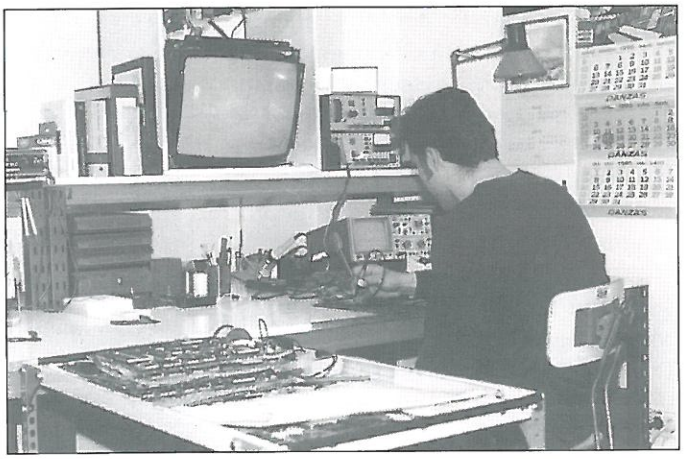
El mercado internacional, aunque muy competitivo, no es un mercado cerrado; esto quiere decir que si una empresa ofrece un producto de calidad, este producto se venderá. En estos momentos, nuestra empresa ofrece unos productos que tienen un nivel de calidad comparable al de cualquier otra empresa del sector, por lo que no tenemos ningún problema para vender en el exterior. De hecho, en torno al 80% de nuestra facturación procede de la exportación; es decir, sin las ventas al exterior Gaelco no podría existir.

En los dos últimos años se ha producido un gran auge de las consolas. Este auge, ¿ha repercutido negativamente en el mercado de las máquinas recreativas?

Si, el boom de las consolas nos ha afectado muy negativamente. Si hace algunos años había 7 grandes compañías en este mercado, actualmente quedan solo Namco, Sega y Williams; y respecto a las demás, a las pequeñas, se puede decir que estamos sobreviviendo. Eso sí, las empresas pequeñas tenemos la ventaja de que, al tener unos gastos estructurales menores, podemos mantenernos con unas ventas también menores, lo que nos permite aguantar la crisis mejor que las compañías grandes.

Dice que hay tres grandes compañías dominando el mercado. ¿Existe un peligro de monopolización? ¿Pueden estos grandes copar el mercado y eliminar a los pequeños?

No creo que exista ese peligro. A mi juicio, el problema no está en las COIN-OP, sino en las consolas; a medio plazo, creo que las consolas desaparecerán y serán sustituidas por la televisión interactiva, y en esta situación sí que es posible que las grandes compañías telefónicas, como la ATT, monopolicen el mercado. Estas grandes compañías no sólo controlan el mercado de las comunicaciones, sino que tienen el poder económico necesario como para poder introducirse en el mercado de los videojuegos, ya sea creando sus propias firmas o comprando otras ya existentes.



El departamento técnico realiza el diseño y la revisión de las placas.

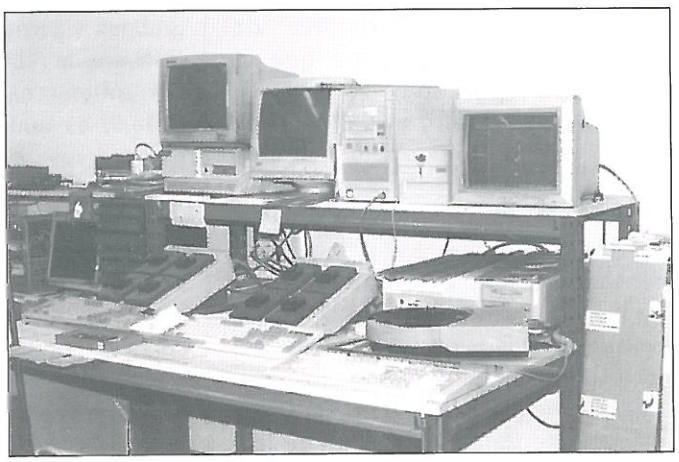
Los juegos son sometidos al banco de pruebas.

Gaelco se dedica a las máquinas de salón, las COIN-OP, pero, ¿tiene previsto ampliar sus actividades a los juegos de ordenador o consola?

Nuestro principal producto es el COIN-OP, que es lo que sabemos hacer; pero eso no quiere decir que despreciemos los juegos de ordenador; la clave es el tipo de juegos que hacemos, nuestros juegos están pensados para COIN-OP, no para ordenador o consola. Para nosotros no sería ningún problema adaptar nuestros juegos a consola; de hecho, las más modernas como la Saturn o la PlayStation, ofrecen unas prestaciones incluso superiores a algunas máquinas de salón, pero para cierto tipo de juegos. Nosotros nos introduciremos en este mercado en el momento en que dispongamos del juego adecuado; de hecho estamos hablando con empresas del ramo y esperamos sacar algo antes de final de año.

¿Qué clase de equipos utilizáis para el desarrollo de vuestros juegos? ¿Cómo programáis las placas?

Para el desarrollo utilizamos emuladores, que son unos aparatos que realizan la función del microprocesador,



Mediante un sofisticado equipo se realiza la grabación de *customs.*

ejecutar programas, pero que además ofrecen la posibilidad de poder controlar esta ejecución; es decir, son debuggers por hardware. Los emuladores se controlan desde un ordenador, en nuestro caso una máquina UNIX, que también se usa para desarrollar el programa.

¿En qué lenguajes desarrolláis los videojuegos?

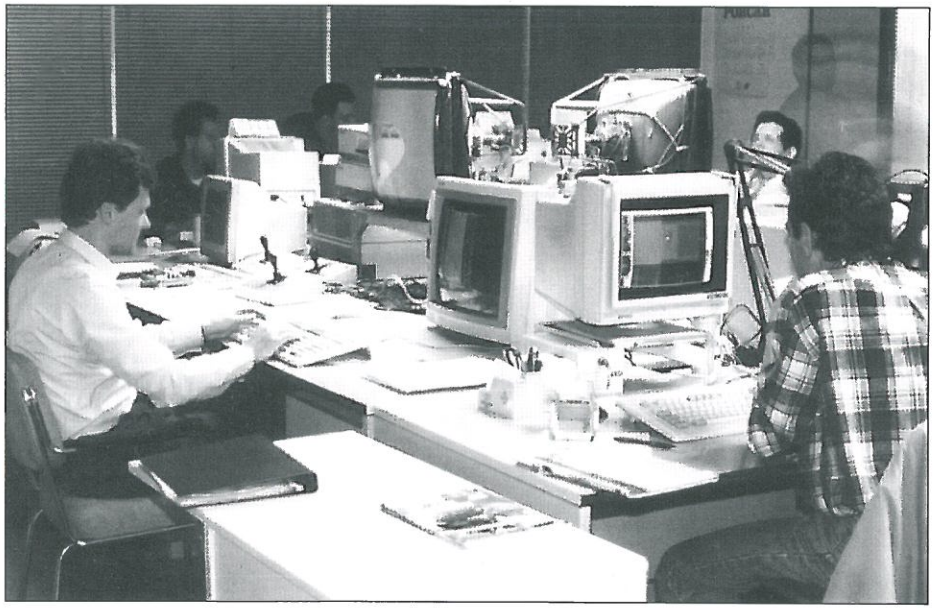
Los juegos se programan en ensamblador; no porque queramos, sino porque es el único lenguaje que permite llevar la máquina al límite de sus posibilidades, cosa necesaria en los juegos. Otra cosa son los programas complementarios, las herramientas auxiliares que se ejecutan en los ordenadores y que se utilizan para generar tablas o para mantenimiento del sistema; estos se programan en lenguajes de alto nivel, como el C o incluso BASIC.

¿Qué se está haciendo ahora en el mercado de los videojuegos?

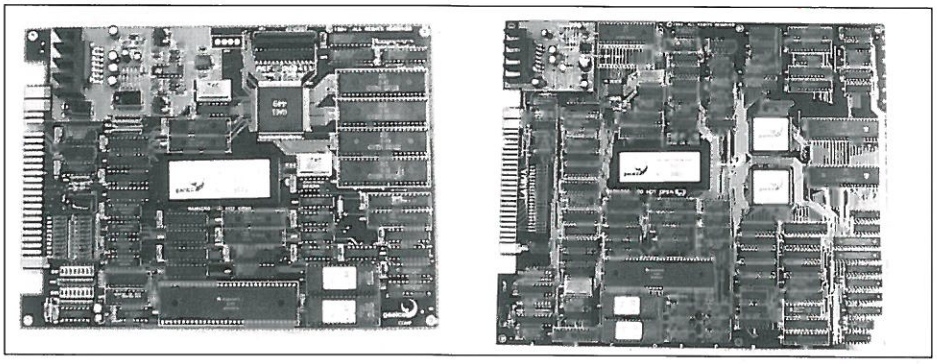
En estos momentos se está produciendo un cambio muy brusco en las tendencias. La gente se ha cansado de los juegos repetitivos de pantallas, los juegos en los que había que vencer grupos de enemigos para pasar a la siguiente pantalla. Actualmente, la gente se decanta más por los juegos deportivos, y los simuladores; un ejemplo son los juegos de fútbol, cada año se hacen cuatro o cinco y todos funcionan; siempre habrá uno que será la estrella, pero todos funcionarán comercialmente.

Antes hemos hablado del auge de las consolas y el declive del COIN-OP. ¿A qué crees que se debe este fenómeno?

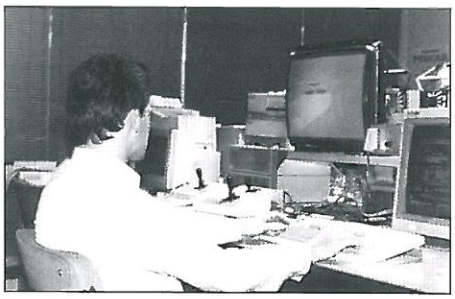
Precisamente a eso, el COIN-OP ha bajado porque las consolas han subido. Es como el vídeo y el cine; el auge del vídeo ha hecho bajar las salas de cine. Además, está el problema del cansancio; la gente se está cansando de los videojuegos, por lo que es necesario cambiar. Actualmente, el sector está en plena reconversión y las salas de juegos clásicas, las salas que únicamente tenían máquinas de juegos están desapareciendo y se están convirtiendo en salas de diversión mucho más general, con una gama de opciones mucho más amplia. Respecto a las consolas, ahí quien realmente ha hecho negocio ha sido Nintendo. El primero fue Atari, pero Nintendo supo corregir los errores de ésta, y lanzar un producto correcto, una consola con una buena calidad de gráficos y sonido, acompañada de un software, el Mario Bross, que ya había sido un gran éxito en COIN-OP. El resultado es que Nintendo, pese a ser una empresa con una estructura mediana, ha alcanzado un peso económico mayor que el de muchos bancos. Tras el éxito de Nintendo, Sega se apresuró a entrar, y aunque también ha tenido mucho éxito, gracias a su agresiva política comercial, no ha llegado a los niveles de Nintendo. El problema es que las empresas del sector de la electrónica de consumo han visto que esto es un gran negocio y también han decidido entrar, con lo que el mercado se ha visto inundado; es decir, las consolas han subido, pero con mucha confusión, porque ahora, al haber tantas empresas, el papá, cuando llegue navidad, no va a saber si comprarla de la misma marca que el televisor o del vídeo, o comprar alguno de los clásicos, que son Sega y Nintendo. Pero el problema también lo tienen las empresas de software, que no saben para que máquinas desarrollar. ■



Los trabajos salen adelante gracias a un equipo de personal cualificado.



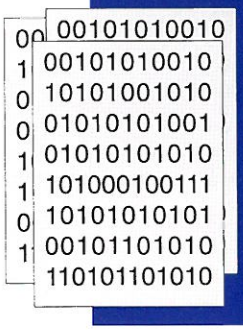
Las placas de juegos son similares a las de los ordenadores personales.



La creación de un programa requiere la utilización de diversos equipos.

CONTROL COMPLETO DE SEGMENTOS

Santos Riancho



En el capítulo anterior se estudiaron las directivas de control simplificado de segmentos. Éstas, como se pudo comprobar, permiten codificar de forma cómoda, clara y, en muchas ocasiones, completa. Por lo tanto su conocimiento es suficiente en la mayoría de ocasiones para llevar a buen término un proyecto. Aún así el estudio de las "antiguas" directivas de control completo de segmento será bastante útil para el neófito, ya que por un lado le permitirá comprender a fondo la gran cantidad de software escrito con su ayuda y, por otro, permitirá un dominio aún mayor sobre la estructura de su programa; en algunas circunstancias esto puede ser muy importante.

En este capítulo se efectuará una breve presentación de cada una de ellas, y, más adelante, se profundizará en sus posibilidades.

DEFINIR UN SEGMENTO

Sin duda alguna, la protagonista de este conjunto de directivas es *SEGMENT*. Mediante ella se construyen los segmentos, se les da un nombre y se les asigna un tipo y un alineamiento, y su sintaxis completa es la mostrada en la figura 1. De su significado y posibilidades nos ocuparemos ahora mismo, pero no sin antes efectuar una puntualización: aunque de forma rigurosa un segmento son 64 Kb consecutivos de memoria, es habitual llamar así a las distintas agrupaciones de datos y/o código delimitadas por las directivas *SEGMENT* y *ENDS* (*END Segment*). Como ya se vio al estudiar las directivas de control simplificado, estas agrupaciones pueden ocupar menos de esa cantidad, pero en cualquier caso se podrán manejar mediante un único valor de un registro de segmento. Esta es la razón de que se emplee dicho término en estas circunstancias.

```
nombre SEGMENT [organización] [READONLY] [combinación] ['clase']
        .....
        Instruciones/Datos
        .....
nombre ENDS
```

Figura 1. Sintaxis para definir un segmento.

NOMBRE DEL SEGMENTO

Es la palabra que precede a la directiva SEGMENT, y debe ser única, salvo en el caso de emplearse en dos o más segmentos que vayan a fundirse en uno sólo en el proceso de enlace. Estos segmentos no necesariamente poseerán los mismos atributos, pero la suma de sus tamaños, obviamente, deberá ser menor de 64 Kb.

Aunque el nombre de un segmento puede ser cualquiera, salvo una palabra reservada, existen algunos preestablecidos. Así, el segmento que va a contener las instrucciones suele ser llamado *_TEXT*, los datos se almacenan en otro llamado *_DATA* y la pila en el denominado *STACK*. Respetar este convenio es recomendable, ya que permite aumentar la reusabilidad de parte o todo el programa.

ORGANIZACIÓN DEL CONTENIDO DE UN SEGMENTO

Cuando ya se ha dado nombre a un segmento se debe indicar a partir de qué punto del programa debe ser situa-

Las directivas de control completo de segmento fueron las primeras con las que contó el lenguaje ensamblador para definir los bloques de código y datos fundamentales de un programa. Su conocimiento no es algo baldío: proporcionan al programador elasticidad y potencia a la hora de programar.

do el primer elemento del mismo. Esto es lo que se denomina organización o alineamiento de un segmento.

Las posibles organizaciones son las mostradas en la figura 2, siendo PARA la opción por defecto, y su significado es el siguiente. Si se establece la organización BYTE, el programa enlazador sitúa el primer componente del mismo a partir de la primera posición que encuentra libre. En caso de tenerse WORD, DWORD, PARA o PAGE el

Organización	Dirección de comienzo
BYTE	Primer byte libre
WORD	Primera palabra libre
DWORD	Primera doble palabra libre
PARA	Primer párrafo libre
PAGE	Primera página libre

Figura 2. Organizaciones posibles para un segmento.

principio del segmento se encontrará, respectivamente, en la primera dirección física disponible múltiplo de 2, 4, 16 ó 256. En este punto es posible que algún lector se muestre extrañado pues ¿no debe comenzar siempre un segmento en una dirección múltiplo de 16? La respuesta a esta pregunta es que, en efecto, un registro de segmento sólo puede apuntar a una posición de estas características, pero el contenido "real" del mismo no tiene porqué someterse a esta ley. En otras palabras, no es preciso que el primer componente de un segmento se sitúe a partir del desplazamiento 0000.

APROVECHAR ESPACIO

El objeto de esto es conseguir, cuando sea necesario, la mayor compactación posible de los datos de un programa. Para verlo mejor supongamos la siguiente situación extrema: se tienen los dos segmentos de la figura 3. El primero de ellos, que supondremos también localizado en primer lugar dentro del archivo ejecutable, comienza en una dirección múltiplo de 16 y su tamaño se reduce a un único byte. Si se emplea para el segundo segmento la organización PARA, al obligarle a comenzar también en una dirección múltiplo de 16 se producirá una situación, en la que se están inutilizando 14 bytes.

Sin embargo, al utilizarse la organización BYTE el segundo segmento se dispondrá de forma que se use el primer hueco libre para alojar el primer dato del mismo. Para ello se tomará como comienzo de segmento la primera posición múltiplo de 16 anterior a la que se ha encontrado, y el desplazamiento asignado a este primer dato será la diferencia entre ambas posiciones. En nuestro ejemplo el inicio del segundo segmento coincidiría con el del primero, y el desplazamiento de su primera variable sería el 0001.

Como ya se dijo, esto permite comprimir al máximo la disposición de los elementos de un programa, pero tiene el inconveniente, muchas veces sin importancia, de disminuir el tamaño máximo de un bloque de datos o instrucciones a menos de 64 Kb.

DATOS PROTEGIDOS

El siguiente atributo, *READONLY*, es opcional y su efecto es el de proteger contra escritura el contenido del bloque definido con la correspondiente directiva *SEGMENT*. Su efecto es más fuerte aún que el de la directiva *.CONST*, vista en el capítulo anterior, ya que no sólo impide la modificación de una variable direccionada de manera directa, sino que fuerza al programa ensamblador a generar un aviso cada vez que en el programa aparecen instrucciones que intentan modificar, mediante referencias directas o indirectas, cualquier zona protegida con este atributo.

Por ejemplo, supóngase que el segmento de datos ha sido creado empleando *READONLY*. Lo normal será que se utilice el registro DS para apuntar a su primera posición y, si repasamos el capítulo dedicado a los modos de direccionamiento, comprobaremos que los registros que permiten generar direcciones tomando como base el valor de DS son SI, DI y BX. Pues bien, si en el programa se tienen, por ejemplo, las instrucciones *MOV [DI], AH*, *ADD [SI+3], DX* o *XOR BYTE PTR [BX], 4Fh*, en el proceso de ensamblado se generará un aviso por cada una de ellas. El programa ensamblador "recuerda" muy bien el empleo habitual de estos registros y pone en guardia al programador sobre cualquier posible desliz.

Además de para proteger datos, este atributo se emplea para evitar sobrescrituras de fragmentos de código. Es destacable que su efecto se extiende a los componentes de todos los segmentos con el mismo nombre que aquel en que ha sido utilizado.

UNIÓN DE SEGMENTOS

Ya se avanzó anteriormente que los segmentos con el mismo nombre eran fusionados por el programa enlazador, que asignaba desplazamientos teniendo en cuenta la posición de cada variable o instrucción dentro del segmento final. El atributo que vamos a tratar ahora se encarga de especificar la forma en que esto se lleva a cabo.

En la figura 4 se muestran las distintas maneras en que se pueden unir segmentos con el mismo nombre. Por defecto la opción seleccionada es *PRIVATE*, que tiene como consecuencia impedir que datos o instrucciones definidos en un módulo puedan ser empleados en otros. Efecto opuesto es el de las opciones *PUBLIC* y *MEMORY*, que pone el contenido del segmento a disposición de otros módulos con el mismo nombre. De todas formas el uso de una u otra no tiene mayor trascendencia cuando se trabaja con segmentos definidos en el mismo programa.

Las directivas de control completo son más flexibles que las de control simplificado

Mayor importancia tiene, a nuestro entender, la opción *STACK*. Como su nombre indica se emplea para crear la pila del programa, y fuerza al enlazador a, una vez agrupados los segmentos del mismo nombre, depositar en la cabecera del programa el tamaño de la pila, con el que se inicializará SP, e información para establecer el comienzo de ésta en el momento de

cargar el programa y poder inicializar SS.

Por otro lado, las consecuencias de la opción *COMMON*, como en el caso de *PUBLIC* y *PRIVATE*, no son visibles a no ser que se trabaje con varios ficheros objeto. La finalidad de esta opción es la de superponer, y no unir, aquellos segmentos que tienen el mismo nombre, de forma que la longitud final del segmento resultante sea la del mayor de los que se agrupan. El programa enlazador calcula los desplazamientos de las variables implicadas a partir del comienzo de dicho segmento final, de forma que a dos o más les puede corresponder el mismo OFFSET. Por este motivo no es conveniente inicializar los campos que pertenecen a segmentos de este tipo, al "machacar" el último campo que llega el contenido de los anteriores. Esta combinación puede utilizarse para manejar variables de gran tamaño que no van a coexistir simultáneamente en memoria.

```
.
.
UNO SEGMENT PARA
      X DB OOh
UNO ENDS

DOS SEGMENT PARA
      Y DB 1024 DUP ('2')
DOS ENDS
.
.
```

Figura 3. Supuesto práctico de dos segmentos.

Por último, la opción *AT dirección* tiene por objeto asignar a un segmento una dirección física concreta. Su mayor utilidad es poder definir dentro de un segmento variables que se corresponden con puntos específicos de la memoria del sistema: tabla de vectores de interrupción, *buffers* de vídeo, el área de datos del BIOS, etc.

TAMAÑO MÁXIMO DE LOS SEGMENTOS

El penúltimo atributo es el que hace referencia al tamaño de los segmentos del programa. Con la opción *USE16* se informa al programa ensamblador de que se manejarán desplazamientos de 16 bits, por lo que el tamaño máximo del segmento en cuestión será de 64 Kb. Mediante *USE32* se permiten desplazamientos de 32 bits, y el tamaño del segmento puede alcanzar los 4 Gb. Evidentemente la última opción sólo es posible en procesadores 386 o superiores.

Llegados a este punto, sólo queda por analizar un atributo, *CLASE*, para dar por finalizado el estudio de la directiva *SEGMENT*. Ésta se emplea para indicar qué segmentos se van a cargar consecutivamente en la memoria y, de igual forma que en el caso del nombre, aunque el programador puede escoger el que prefiera, existen algunos predefinidos, que son los mostrados en la figura 5. Dada su finalidad, las posibilidades de este atributo se abordarán a la vez que se analiza un problema que comenzamos a plantearnos: el del orden de los segmentos dentro del archivo ejecutable.

DISPOSICIÓN DE SEGMENTOS

En la mayoría de ocasiones no es imprescindible saber cómo se sitúan realmente los elementos de un programa dentro del mismo. Sin embargo, puede ocurrir que para alguna actividad específica sea necesario conocer la localización de cierto tipo de datos o, por ejemplo, la dirección de la última instrucción del programa. En este caso será preciso tener muy claros los pasos que ha seguido el programa enlazador hasta obtener el archivo ejecutable.

En principio los segmentos se sitúan en memoria por orden de aparición. Es decir, se asignan las primeras direcciones a aquellos segmentos que tienen el mismo nombre que el detectado en primer lugar. Lo normal es que los segmentos del mismo nombre posean también la misma clase, pero de no ser así se produciría una clasificación por clase dentro del segmento.

Otra opción es la de ordenar alfabéticamente los segmentos por su nombre y, por último, ordenarlos según el convenio del sistema operativo DOS.

Este convenio supone lo siguiente: en primer lugar se colocan los segmentos cuya clase es *CODE* y a continuación aquellos segmentos de datos cuya clase no es *BSS*, o sea, aquellos que pueden tener un valor inicial; después se sitúan los segmentos de clase *BSS* y al final los destinados a pila con clase *STACK*.

Para escoger un tipo u otro de ordenación basta con situar una de las directivas al comienzo del programa. La opción por defecto es la secuencial, por lo que el uso de la directiva *.SEQ* se puede omitir.

GRUPOS DE SEGMENTOS

Hasta el momento se ha hablado de definir segmentos y enlazarlos por separado, esto es, asignando desplazamientos de forma aislada a las instrucciones y datos de aquellos que poseían el mismo nombre o bien pertenecían a la misma clase.

Esto es suficiente cuando el tamaño conjunto de todos ellos es mayor que el de un segmento físico, pero puede darse el caso de que dos o más segmentos distintos ocupen menos de 64 Kb en modo 16 bits o 4 Gb en modo 32 bits. Si fuera así sería un verdadero "derroche" enlazar estos segmentos por separado y obtener desplazamientos con respecto a distintos puntos, ya que habría que emplear varios registros de segmento donde con uno bastaría.

Mediante SEGMENT y ENDS se delimita el contenido de un segmento

Este problema se puede solucionar con la directiva *GROUP*. nombre GROUP segmento_l[, segmento_2]... Mediante su empleo es posible asignar a todos los segmentos declarados a la derecha de la directiva una única dirección de comienzo, representada por el nombre del grupo.

Un posible uso inmediato de *GROUP* es el de situar en el mismo segmento físico los datos y la pila de un programa. De hecho, esto es parte de la misión de la directiva de control simplificado *.MODEL*, vista en el anterior

capítulo: salvo cuando se especifica la opción *FARSTACK*, se crea un grupo denominado DGROUP en el que tienen cabida los datos inicializados y sin inicializar del programa, así como su *STACK*. En el caso de emplearse la opción antes indicada, DGROUP queda constituido tan sólo por los datos.

En este momento es conveniente ilustrar mediante ejemplos los conceptos vistos en este capítulo. Para ello se dispone de varios archivos fuente incluidos en el disquete que acompaña a este número de la revista. Sin embargo, antes de comentar el contenido de cada uno de estos archivos queda un tema pendiente: el de cómo relacionar los contenidos de los segmentos definidos en el programa con los registros de segmento del procesador en algunas circunstancias críticas. Esto se hace con ayuda de la directiva *ASSUME*.

El contenido de un segmento puede comenzar en una dirección NO múltiplo de 16

LA DIRECTIVA *ASSUME*

Esta directiva, nada espectacular en apariencia, tiene, entre otros, los formatos mostrados en la figura 9.

Mediante el primero de ellos se consigue que el ensamblador tenga en cuenta el segmento en que está definida una variable o etiqueta cuando éstas se emplean en una instrucción de transferencia indirecta de control. ¿Qué significa esto? Supóngase que dentro del segmento de código se define la variable DIRECCION DWORD, y que esa variable almacena la dirección física de la primera instrucción de una subrutina *FAR*. Si ahora se desea invocar esa subrutina será necesaria la instrucción *CALL CS:DIRECCION* para transferir el control a ese punto del programa, ya que, de otra forma, el programa ensamblador entenderá que el desplazamiento de DIRECCION ha de ser calculado con respecto al inicio del segmento de datos.

Sin embargo, si antes de esa instrucción se ha codificado *ASSUME CS:Segmento_de_código*, el programa ensamblador tendrá muy claro, sin necesidad del uso del prefijo de segmento, que la referencia de la variable es con respecto al inicio del segmento de código, por lo que bastará con *escribir CALL DIRECCION* para que se produzca el resultado deseado. Evidentemente es responsabilidad del programador el que cada registro de segmento almacene el valor adecuado en cada momento.

DESHACER LA ASOCIACIÓN

Utilizando la palabra reservada *NOTHING* se elimina la asociación existente entre un registro de segmento y el contenido del segmento asociado a él mediante una directiva *ASSUME* anterior. Por ejemplo, con *ASSUME CS:NOTHING* después de la instrucción *CALL DIRECCION*, el ensamblador dejaría de poner por su cuenta el prefijo de segmento antes de la variable, en caso de volver a invocarse la subrutina.

LOS PROGRAMAS EJEMPLO

Con esta información adicional ya estamos en condiciones de echar un vistazo a los programas ejemplo.

Los dos primeros, COM1.ASM y COM2.ASM muestran cómo construir un ejecutable con extensión COM. Obsérvese que en ambos casos basta con definir un único segmento, _TEXT, de clase CODE. El nombre podría haber sido otro, por ejemplo CODIGO, pero es necesario que la clase sea de ese tipo en concreto o de lo contrario el ensamblador lanzará un mensaje de error. Dado que en el programa no aparece ninguna llamada indirecta, no habría sido necesario emplear la directiva ASSUME; sin embargo se ha dispuesto de esta forma para ilustrar su uso. Otro punto en común entre los dos archivos es el empleo de la directiva *ORG número*.

CÁLCULO DE DESPLAZAMIENTOS

Éste se emplea para indicar al enlazador que debe añadir la cantidad 0100h a todos y cada uno de los desplazamientos obtenidos al procesar el código fuente. Ya vimos al estudiar el control simplificado de segmento que, en un programa COM, había que tener en cuenta el espacio ocupado por el PSP. En aquella ocasión era la directiva *.STARTUP* la encargada de recordar al enlazador esta necesidad, pero manejando este otro conjunto de directivas hay que indicarlo explícitamente.

MEZCLA DE CÓDIGO Y DATOS

Otro aspecto típico de los archivos COM se pone de manifiesto en ambos ejemplos: datos, código y pila se encuentran "mezclados". También esto se vio en el capítulo anterior, pero el empleo de las directivas *.DATA*, *.DATA?*, *.CONST* y *.CODE* daban un aspecto más claro al fuente, aunque sabíamos, claro está, que la forma final del programa era como la que se puede apreciar en COM1.ASM y COM2.ASM.

La única diferencia entre estos dos

Combinación	Acción con otros módulos
PRIVATE	No combinación
PUBLIC	Combinación
MEMORY	Combinación
COMMON	Superposición
AT dirección	Asignación de dirección de inicio de segmento

Figura 4. Formas de unir segmentos con el mismo nombre.

archivos es la referente a la situación relativa de datos e instrucciones. En COM1.ASM se trabaja a partir del hecho de que el sistema operativo inicializa IP con el valor 0100h al ejecutar el programa, por lo que los datos se sitúan detrás de las instrucciones. En el caso de COM2.ASM se ha optado por la forma habitual de un programa lanzador de residentes: como primera instrucción un salto por encima de los datos y/o instrucciones que no formen parte del programa principal, a continuación aquellos, y por último el código del programa en si.

ARCHIVOS EXE

Como ejemplos de codificación elemental de ejecutables con extensión EXE se tienen EXE.ASM y GRPEXE.ASM.

El primero muestra una forma sencilla de obtener un ejecutable con zonas aisladas de datos inicializados y sin inicializar, una pila de 1 Kb y algunas instrucciones. Más que comentar el cometido de cada directiva, lo que conviene aquí es que el lector ensamble el programa y estudie el ejecutable con ayuda de, por ejemplo, el comando DEBUG del DOS. Modificando o suprimiendo las distintas directivas podrá comprobar si las variaciones del archivo fuente se traducen en cambios substanciales del ejecutable o si, por el contrario, no tienen ningún efecto o producen error.

READONLY protege datos y código contra escritura

El segundo enseña cómo se puede crear un grupo que albergue todo tipo de datos del programa. Quizá lo más destacable sea el hecho de que ha sido necesario incluir tras la inicialización de DS (que también ha sido efectuada en el ejemplo anterior) un bloque de instrucciones destinado a modificar SS y SP de manera que la pila y los datos formen parte del mismo segmento físico sin superponerse. Cuando se usaban las directivas de control simplificado de segmento, todas estas instrucciones eran generadas por el ensamblador al expandir la directiva *.STARTUP* cuando el tipo de pila escogido era *NEARSTACK*.

Nombre	Clase	Uso
FAR_BBS	FAR_BBS	Datos FAR sin inicializar
FAR_DATA	FAR_DATA	Datos FAR inicializados
STACK	STACK	Pila
_BBS	BBS	Datos NEAR sin inicializar
_DATA	DATA	Datos NEAR inicializados
_TEXT	CODE	Instrucciones

Figura 5. Nombres predefinidos por el artículo CLASE.

Evidentemente no hay ninguna restricción a la hora de definir más variables tras esta última parte del programa, pero, como se suele decir, "aunque todo puede hacerse no todo debe hacerse". Salvo casos excepcionales que exijan lo contrario los datos, las instrucciones y la pila de un programa deben tener bien delimitadas sus zonas dentro del fuente y el ejecutable, aun con riesgo de encarecer sin ningún provecho la codificación y el mantenimiento del programa.

La directiva GROUP permite agrupar en un segmento físico, información de diverso tipo

```
ASSUME reg_segmento_1:nombre_segmento [, reg_segmento_2:...]...
ASSUME reg_segmento_1:NOTHING [, reg_segmento_2:NOTHING]...
```

Figura 6. Formatos de la directiva *ASSUME*.

PARA FINALIZAR

En este capítulo se ha tenido una primera toma de contacto con las directivas de control completo de segmento. Sin embargo, como ya se indicó al principio, sólo hemos vislumbrado algunas de sus posibilidades. Su potencia real se pone de manifiesto cuando se trabaja con un proyecto que implica el uso de varios ficheros objeto y/o librerías.

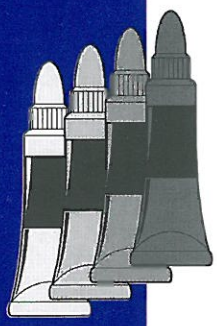
Como se ha podido comprobar, su sintaxis y manejo son más complicados que en el caso de las directivas estudiadas en el capítulo anterior. Esto, como era de esperar, se traduce también en una mayor capacidad de maniobra al codificar y una mayor eficiencia a la hora de obtener el archivo ejecutable. ■

BIBLIOGRAFÍA

- Microsoft Macro Assembler Programmer's Guide-, V 6.0 y sucesivas, Microsoft Corporation
- Microsoft Macro Assembler Reference, Microsoft Corporation
- Michael Tischer: PC interno 2.0. Programación del sistema. DATA BECKER, 1994
- Angulo-Funke:386 y 486-Microprocesadores avanzados de 32 bits. EDITORIAL PARANINFO
- Scanlon, Leo J.:-80286, Programación ensamblador en entorno MS-DOS-. EDICIONES ANAYA MULTIMEDIA, Madrid, 1988.
- Harley Hahn, -Assembler Inside & Out-, Osborne McGraw-Hill 1992
- Miguel Angel Rodríguez-Roselló, - 8088-8086/8087 Programación Ensamblador en entorno MS-DOS-, EDICIONES ANAYA MULTIMEDIA, Madrid, 1988
- Jürgen Hegner, -8086/8088 Programación en ensamblador-, Ed. DATANET, Barcelona

NORMA VESA

Agustín Guillén



Con la base de la clásica tarjeta gráfica VGA, muchos fabricantes desarrollaron sus propias SVGA, de tal manera que eran totalmente incompatibles unas con otras. Esto ocasionaba una dificultad enorme en la realización de aplicaciones y programas que necesitaban de estos modos gráficos mejorados. Para solventar este problema, se creó la norma VESA y con ello se simplificó la programación de las complejas SVGA.

Básicamente se trata de una ampliación de la BIOS de vídeo, manejada a través de la interrupción 10h, y su objetivo es, por un lado, el de permitir obtener información sobre las posibilidades de una determinada tarjeta SVGA, y por otro, el de programar el hardware de dicha tarjeta de una manera totalmente estándar. Hasta hace poco (y todavía es el sistema para muchas tarjetas de vídeo un poco antiguas) el modo de obtener la compatibilidad VESA era el de cargar un *driver* o programa residente, que se encargaba de "capturar" las llamadas a la norma VESA y las emulaba. De esta manera, muchos juegos o programas gráficos vienen acompañados de un directorio repleto de *drivers* VESA, uno para cada tipo de tarjeta gráfica existente. También existe un programa "multi-tarjeta", habitualmente llamado Univesa que detecta el tipo de SVGA conectada e instala unas u otras rutinas emuladoras. En la actualidad, la práctica totalidad de tarjetas de vídeo que se comercializan, incorporan la norma VESA en sus BIOS, librando al usuario del engorro de cargar el *driver* emulador.

DESCRIPCIÓN

La versión más extendida en la actualidad es la 1.2 y será la que se describa en este artículo. Incluso en el disco que acompaña a la revista, se incluye la descripción oficial de esta versión.

Los modos gráficos VESA comienzan a partir del número 100h (15 bits). La causa de esto es que desde el modo 00h al 13h pertenecen a la norma VGA estándar, desde el 14h al 7fh han sido utilizados por los distintos fabricantes para sus modo gráficos SVGA antes de la norma VESA y los comprendidos entre el 80h y el ffh, no están disponibles al tener el bit 7 activado (la rutina clásica de la BIOS de vídeo para cambiar de modo gráfico, 00h, interpreta el bit 7 como *flag* para borrar la pantalla).

En la figura 1 se pueden observar los modos gráficos definidos bajo VESA, nótese el detalle de que comienzan allí donde acaban los modos estándares definidos en VGA (respecto a sus resoluciones). Obviamente no todas las tarjetas gráficas disponen de todos los modos VESA, pues algunos requieren una tarjeta true color o la posesión de dos megabytes de memoria de vídeo.

BIOS VESA o EXTENDED VGA BIOS

Se compone tan sólo de 8 funciones, pero con ellas es posible realizar cualquier acción sobre los modos extendidos. Esta nueva BIOS no incorpora ninguna función de dibujo o de salida a pantalla, se limita tan sólo a informar del hardware conectado y a interaccionar con él. Incluso estas funciones están agrupadas en una sola llamada, con el objetivo de no interferir en las anteriores versiones de BIOS.

El acceso es mediante la interrupción normal de acceso al vídeo, la INT 10h y la función 4fh indicada en el registro AH, quedando el número de subfunción en el registro AL.

En el siguiente ejemplo se observa claramente:

```
mov al, 00h ; Pide la información SVGA
mov ah,4fh ; Llamada a la BIOS VESA
int 10h ; BIOS de vídeo
```

Todas las funciones retornan en el registro AX la información sobre el resultado de la llamada. En el byte bajo (AL), indican si la función es soportada mediante el valor 4fh y en el byte alto (AH) si la llamada ha sido correcta con

el valor 00h o si algo ha fallado con el valor 01h.

Las funciones VESA son las siguientes:

Petición de información SVGA (00h)

Informa de las capacidades generales de la tarjeta SVGA conectada al equipo. El traspaso de información se realiza mediante un *buffer* que es rellenado por la función.

Entradas: AH = 4Fh Llamada a una función VESA
AL = 00h Número de subfunción VESA
ES:DI = Puntero al buffer

Salidas: AX = Estado

El buffer o estructura tiene el siguiente formato:

```
VgaInfoBlock   STRUC
VESASignature      db          'VESA'
; Cadena "VESA"
VESAVersion        dw          ?
; Número de versión VESA
OEMStringPtr       dd          ?
; Puntero al nombre del fabricante
Capabilities       db          4 dup(?)
; Posibilidades del entorno de vídeo
VideoModePtr       dd          ?
; Puntero a los modos soportados
TotalMemory        dw          ?
; Número bloques de 64 kb en tarjeta
Reserved           db          236 dup(?)
; Resto del bloque
VgaInfoBlock   ENDS
```

El campo *Capabilities* define mediante bits las distintas capacidades del equipo, la definición de los bits es la siguiente:
B0 0 El DAC es de anchura fija, con 6 bits por color primario
1 La anchura del DAC es modificable
B1-B31 Reservados

La lista de los modos de vídeo soportados se compone de *words*, una por modo, y termina con el valor 0ffffh (-1).

Petición de información sobre un modo SVGA (01h)

Para conocer más características sobre un determinado modo de vídeo (siempre dentro de la lista de modos mostrada por la función 00h), se utiliza la función 01h. La información, al igual que la función anterior, se rellena en un buffer o estructura, cuya dirección a elegido el propio programador.

Entradas: AH = 4Fh
AL = 01h
CX = Modo gráfico solicitado
ES:DI = Puntero a un buffer de 256 bytes

Salidas: AX = Estado

El buffer tiene el formato descrito a continuación:

```
ModeInfoBlock  STRUC
; Campos obligatorios
ModeAttributes dw  ? ; Atributos del modo
WinAAttributes db  ? ; Atributos de la ventana A
WinBAttributes db  ? ; Atributos de la ventana B
WinGranularity dw  ? ; Granularidad de ventana
WinSize    dw  ?     ; Tamaño de ventana
WinASegment dw  ?    ; Segmento inicial de la
                     ventana A
WinBSegment dw  ?    ; Segmento inicial de la
                     ventana B
WinFuncPtr dd  ? ; Puntero a la función de ventanas
BytesPerScanLine dw  ? ; Bytes por línea de scan

; Campos opcionales (antes de la versión 1.2,
ahora obligatorios)
XResolution  dw  ?   ; Resolución horizontal
YResolution dw  ?    ; Resolución vertical
XCharSize db  ?      ; Anchura del carácter
YCharSize db  ?      ; Altura del carácter
NumberOfPlanes     db  ?
; Número de planos de memoria
BitsPerPixel db  ?   ; Bits por pixel
NumberOfBanks db  ? ; Número de bancos
MemoryModel db  ?    ; Tipo de modelo de memoria
BankSize  db  ?      ; Tamaño del banco en Kb
NumberOfImagePages db  ?; Número de imágenes
Reserved           db  1; Reservado para la
                     función de paginación

; Campos de Color Directo (obligatorios si
;MemoryModel es 6 o 7)
MaskSize db  ?
; Tamaño de máscara del color rojo (bits)
RedFieldPosition db  ?
; Posición LSB del bit de la máscara roja
GreenMaskSize db  ?
; Tamaño de máscara del color verde (bits)
GreenFieldPosition db  ?
; Posición LSB del bit de la máscara verde
BlueMaskSize db  ?
; Tamaño de  máscara del color azul (bits)
BlueFieldPosition db  ?
; Posición LSB del bit de la máscara azul
RsvdMaskSize db  ?
; Tamaño de  máscara del color reservado
DirectColorModeInfo  db  ?
; Atributos de modo de Color Directo
Reserved db  216 dup(?) ; Resto del buffer
ModeInfoBlock  ENDS
```

El campo *ModeAttributes* describe las características más importantes del modo solicitado. Su definición es la siguiente:
B0 0 = Modo no soportado por el hardware
1 = Modo soportado por el hardware
B1 = 1 (Reservado)
B2 0 = Funciones de salida no soportadas por la BIOS
1 = Funciones de salida soportadas por la BIOS
B3 0 = Modo monocromo
1 = Modo color
B4 = Tipo de modo
0 = Modo de texto
1 = Modo gráfico
B5-B15 = Reservado

Los campos *WinAAttributes* y *WinBAttributes* describen las características del funcionamiento de las ventanas:
B0 0 = Ventanas no soportadas
1 = Ventanas soportadas
B1 0 = Ventana de no lectura
1 = Ventana de lectura
B2 0 = Ventana de no escritura
1 = Ventana de escritura
B3-B7 = Reservados

Si las ventanas no son soportadas, se considera que la memoria de vídeo se encuentra en las direcciones habituales, siempre teniendo en cuenta el modo de memoria (MemoryModel) activo.

El campo *WinGranularity* especifica la unidad mínima, en Kb, en la que las ventanas pueden ser reubicadas.

En el campo *MemoryModel* se indica la organización de memoria utilizada en el modo solicitado. La lista de modos es la siguiente:
00h Modo texto
01h CGA
02h Hércules
03h Planar de 4 planos
04h *Pixel* empaquetado
05h 256 colores, 4 no encadenados
06h Color directo
07h YUV
08h-0Fh
Reservados para VESA
10h-FFh Reservados para fabricantes OEM

NumberOfBanks especifica el número de bancos en las que las líneas de *scan* están agrupadas. En los modos que no tienen bancos (los clásicos modos de memoria lineal), este campo debe estar a 1. Si el modo dispone de bancos, su tama–o estará indicado en el campo *BankSize*, en unidades de 1 Kb.

Los campos *RedMaskSize*, *GreenMaskSize*, *BlueMaskSize* y

RsvdMaskSize definen el tamaño, en bits, de los componentes de color existentes en el modo de color directo. En el modo YUV, el valor V se almacena en el campo rojo, el Y en el verde y el U en el azul.

El campo *DirectColorModeInfo* describe importantes características del modo de color directo:

B0 0 = La rampa de color es fija
1 = La rampa de color es programable
B1 0 =Bits en el campo Rsvd son reservados
1 = Bits en el campo Rsvd son utilizables por la aplicación

Activar modo SVGA (02h)
Inicializa un modo gráfico.

Entradas:AH = 4Fh
AL = 02h
BX = Modo de vídeo
B0-B14 = Modo
B15 = Flag de limpiar memoria
0 = Limpiar memoria de vídeo
1 = No limpiar memoria de vídeo

Salidas: AX = Estado

Retorno del modo de vídeo activo (03h)
Retorna el modo de vídeo actual.

Entradas: AH = 4Fh
AL = 03h

Salidas: AX = Estado
BX = Modo actual

Salvar o restaurar el estado de vídeo SVGA(04h)
Mediante esta función se puede salvar o restaurar el estado completo de la SVGA. Si en el registro de máscara (CX) se indica el valor 000fh, todo el estado del hardware de vídeo será salvado (menos la memoria de vídeo, claro).

Entradas:AH = 4Fh
AL = 04h
DL = 00h Retorna el tamaño del buffer para salvar o restaurar
CX = Estados solicitados
B0 = Salvar/Restaurar el estado del hardware de vídeo
B1 = Salvar/Restaurar el estado de la BIOS
B2 = Salvar/Restaurar el estado del DAC
B3 = Salvar/Restaurar el estado de la SVGA

Salidas: AX = Estado
BX = número de bloques de 64 bytes necesarios para guardar el estado

Entradas:AH = 4Fh
AL = 04h
DL = 01h Salvar el estado de la SVGA
CX = Estados pedidos
ES:BX = Puntero al buffer

Salidas: AX = Estado

Entradas:AH = 4Fh
AL = 04h
DL = 02h Restaurar el estado de la SVGA
CX = Estados pedidos
ES:BX = Puntero al buffer

Salidas: AX = Estado

Control de la ventana de la memoria de vídeo (05h)
Esta función fija u obtiene la posición de una ventana especificada dentro de la memoria de vídeo. Antes de utilizar esta función se deben conocer las características del modo de vídeo activo, tales como el tamaño, posición y granularidad de las ventanas (función 01h). Para optimizar el rendimiento en el cambio de ventanas, esta función también se puede ejecutar directamente mediante una llamada larga, basada en la dirección almacenada en el campo *WinFuncPtr* de la estructura *ModeInfoBlock*, rellenada mediante la función 01h. Las ventajas de esto son evidentes, al ser mucho más rápida una simple llamada (CALL) que una lenta y compleja interrupción (INT).

Entradas:AH = 4Fh
AL = 05h
BH = 00h Seleccionar ventana en la memoria SVGA
BL = Número de ventana
0 = Ventana A
1 = Ventana B
DX = Posición de la ventana en la memoria de vídeo

Salidas: AX = Estado

Entradas:AH = 4Fh
AL = 05h
BH = 01h Retornar posición de la ventana en memoria SVGA
BL = Número de ventana
0 = Ventana A
1 = Ventana B

Salidas: AX = Estado
DX = Posición de la ventana

Cambiar u obtener la longitud lógica de una línea de scan (06h)
Cambia o informa de la longitud de la línea de scan. Mediante esta función se puede modificar la memoria de vídeo para que sea más ancha que la zona visualizada. Esta rutina también está disponible en los modos de texto.

Entradas:AH = 4Fh
AL = 06h
BL = 00h Longitud de la línea de scan
CX = Anchura en pixels
Salidas: AX = Estado
BX = Bytes por línea de scan
CX = Pixels por línea de scan
DX = Número máximo de líneas de scan

Entradas:AH = 4Fh
AL = 06h

En la actualidad la práctica totalidad de tarjetas de vídeo que se comercializan incorporan la norma VESA en sus BIOS

BL = 01h Retornar longitud de la línea de scan

Salidas: AX = Estado
BX = Bytes por línea de scan
CX = Pixels por línea de scan
DX = Número máximo de líneas de scan

Cambiar u obtener el inicio de la porción visualizada (07h)

Selecciona el *pixel* a ser visualizado en la esquina superior izquierda de la pantalla. Con esta función se obtienen efectos de *scroll* al poder ir cambiando el inicio visualizado dentro de una pantalla lógica mayor. También es utilizable para el intercambio entre dos distintas pantallas y así conseguir animaciones suaves. Así mismo, esta rutina está disponible en los modos de texto.

Entradas:AH = 4Fh
AL = 07h
BH = 00h Reservado (0)
BL = 00h Inicio de la pantalla
CX = Primer *pixel* visualizado en la línea de scan
DX = Primera línea de *scan* visualizada

Salidas: AX = Estado

Entradas:AH = 4Fh
AL = 07h
BL = 01h Retorna inicio de pantalla

BL = 00h Selecciona la anchura del DAC de paleta
BH = Número de bits de color primario deseado

Salidas: AX = Estado
BH = Número de bits de color actual.

Entradas:AH = 4Fh
AL = 08h
BL = 01h Solicita la anchura del DAC de paleta

Salidas: AX = Estado
BH = Número de bits de color actual

Si desea más información sobre la norma VESA, puede solicitarla a:

Video Electronic Standards Association
Suite 360
2150 North First Street
San Jose, CA 95151-2020

Tel: 408-435-0333
Fax: 408-435-8225

EJEMPLO PRÁCTICO

Se ha desarrollado un sencillo ejemplo en lenguaje ensamblador para mostrar el uso práctico de las funciones VESA. Como se puede observar, el uso de la norma VESA es muy sencillo y se ha pasado de la complejidad más dura en la programación de tarjetas SVGA a una facilidad pasmosa en su utilización. Los fuentes del programa de ejemplo se encuentran en el disco que se adjunta con la revista.

Siempre que se trabaje con VESA se deben definir en primer lugar los *buffers* para almacenar los datos sobre la versión VESA instalada en el equipo, así como las características técnicas del modo o modos que nos interesen.

Siempre se debe comprobar (mediante la función 00h) qué la tarjeta de vídeo instalada dispone de la extensión de la BIOS VESA. Además, por medida preventiva, conviene comprobar que en los 4 primeros bytes del buffer VESA se encuentra la cadena ASCII "VESA".

MODOS GRÁFICOS

Número de modo 15 bits	Número de modo 7 bits	Resolución	Colores	Planos de color
100h	-	640x400	256	
101h	-	640x480	256	
102h	6Ah	800x600	16	
103h	-	800x600	256	
104h	-	1024x768	16	
105h	-	1024x768	256	
106h	-	1280x1024	16	
107h	-	1280x1024	256	
10Dh	-	320x200	32K	(1:5:5:5)
10Eh	-	320x200	64K	(5:6:5)
10Fh	-	320x200	16.8M	(8:8:8)
110h	-	640x480	32K	(1:5:5:5)
111h	-	640x480	64K	(5:6:5)
112h	-	640x480	16.8M	(8:8:8)
113h	-	800x600	32K	(1:5:5:5)
114h	-	800x600	64K	(5:6:5)
115h	-	800x600	16.8M	(8:8:8)
116h	-	1024x768	32K	(1:5:5:5)
117h	-	1024x768	64K	(5:6:5)
118h	-	1024x768	16.8M	(8:8:8)
119h	-	1280x1024	32K	(1:5:5:5)
11Ah	-	1280x1024	64K	(5:6:5)
11Bh	-	1280x1024	16.8M	(8:8:8)

MODOS DE TEXTO

Número de modo 15 bits	Número de modo 7 bits	Columnas	Filas
108h	-	80	60
109h	-	132	25
10Ah	-	132	43
10Bh	-	132	50
10Ch	-	132	60

Una vez localizada la norma, se buscará el modo gráfico que nos interese mediante la lista de modos devuelta en el campo *VideoModePtr*, y se solicitará información adicional (función 01h) sobre dicho modo. Con la comprobación del bit 0 del campo *ModeAttributes*, se obtendrá la más absoluta certeza de que el modo en cuestión está disponible para ser utilizado. Ya sólo falta activar el modo mediante la función 02h y empezar a trabajar con él. Antes de la activación del modo, conviene anotar el modo original (función 03h) y salvar toda su situación y estados (función 04h) para dejar el sistema como estaba antes de empezar. ■

La BIOS VESA se compone tan sólo de 8 funciones

Salidas: AX = Estado
BH = 00h Reservado (0)
CX = Primer *pixel* visualizado en la línea de *scan*
DX = Primera línea de scan visualizada

Cambiar u obtener el modo de control del DAC de la paleta (08h)

Algunos DACs son configurables y permiten 6, 8 o más bits para definir un color primario (R, G o B).

Entradas:AH = 4Fh
AL = 08h

DISEÑO DESCENDENTE

José C. Remiro

La programación modular se basa en que es más fácil escribir un buen programa si éste se divide en partes más manejables. Poder comprobar cada uno de los módulos independientemente, junto con la reutilización del código ya construido, son razones de peso para utilizar la programación modular.

La técnica más utilizada para la construcción de programas complejos, es la subdivisión de éstos en subprogramas más sencillos, de tal forma que uniéndolos se obtenga el programa a construir. Este proceso de subdivisión de programas puede realizarse a su vez con los subprogramas, con el fin de obtener subprogramas suficientemente sencillos como para construirlos y comprobarlos independientemente, combinarlos adecuadamente y obtener el programa global.

Al anterior proceso se le denomina técnica "divide y vencerás", "diseño descendente" (top down) o "refinamiento sucesivo". En este artículo se hablará de las herramientas de que disponen los lenguajes procedurales de alto nivel, para la definición de procedimientos y funciones, que son los elementos del lenguaje que permiten utilizar el diseño descendente.

Las ventajas de dividir un programa en subprogramas o módulos son las siguientes:

1.- El programa al estar dividido en procesos más pequeños, es más manejable, por lo que se facilita su implementación y validación.
2.- El subprograma construido puede ser utilizado en diferentes puntos del mismo programa.
3.- No es necesario construir código de subprogramas que se han codificado, por lo que se ahorra tiempo de programación.
4.- La localización de un error en un programa es más fácil, éste ha sido subdivido en subprogramas, y estos se han validado con anterioridad.

FUNCIONES

Las funciones de un lenguaje de programación siguen la misma pauta asociada a las funciones matemáticas: éstas admiten una serie de valores que en programación denominamos parámetros, en sus variables y nos devuelven un resultado.

Se aumentará el pseudocódigo que se está utilizando hasta ahora para poder declarar funciones. Una declaración de función se compone de la palabra función (que será una palabra reservada en pseudocódigo), seguido de un nombre para la misma, que será un nombre de identificador válido, y a continuación, y entre paréntesis, los parámetros que aceptará la función separados por comas (los nombres de parámetros serán igualmente identificadores válidos), todos estos elementos se denominan cabecera de la función. A continuación, se podrá incluir declaraciones de variables, que pertenecerán a la función (más adelante se explicarán las restricciones de uso de estas variables) y a continuación el conjunto de instrucciones que forman el cuerpo de la función, entre las palabras reservadas inicio y fin.

La función *factorial* del cuadro 1, admite como parámetros un número entero, representado por x. Para su uso, dispone de las variables enteras acumulador y contador. El conjunto de instrucciones que pertenecen a la función se encuentran situadas entre el inicio y el fin más externos. Se puede comprobar que esta función, dado un número entero positivo calcula su factorial (factorial(n)= n·(n-1)·...·3·2·1). Hay que hacer notar que el valor devuelto por la función, será aquel que se le asigne a ésta (en nuestro caso factorial).

La llamada a una función se realiza siempre dentro de una expresión, siendo el valor devuelto por la función el que corresponda al sustituir los parámetros formales de la declaración de la función por los valores dados en la llamada a la función. Los parámetros formales pueden ser sustituidos tanto por literales, como por variables, en cuyo caso, el valor que se tomará será el contenido de la variable en el momento de la llamada a la función.

Es obligatorio en todos los lenguajes de programación, incluir en la declaración de una función el tipo de los parámetros de la función, en el pseudocódigo se hará de igual forma.

PROCEDIMIENTOS

Una de las deficiencias de las funciones, es su incapacidad para devolver múltiples valores de manera simultánea. Si en la función *factorial*, se hubiese deseado devolver un valor de tipo lógico indicando que el parámetro introducido no era un número positivo, no se hubiese podido realizar pues, o bien la función devuelve el valor verdadero o falso dependiendo del éxito de la función, o bien la función devuelve el correspondiente cálculo del factorial del número introducido como parámetro.

La idea de procedimiento es similar a la de función. De hecho, la declaración de un procedimiento coincide con la de función, excepto en que el nombre del procedimiento no lleva asociado ningún valor. Las transformaciones realizadas por un procedimiento en un programa se realizarán a través de sus parámetros.

Se incluye en el pseudocódigo utilizado hasta ahora, un nuevo elemento para poder declarar procedimientos. La declaración de un procedimiento comienza por la palabra reservada procedimiento, seguido por el nombre de procedimiento (cualquier identificador válido) y a continuación, entre paréntesis y separados por coma, los parámetros del procedimiento. Tras la cabecera del procedimiento, podemos incluir variables propias del procedimiento y el conjunto de instrucciones que pertenecen al procedimiento, entre las palabras reservadas *INICIO* y *FIN*.

La llamada a un procedimiento se puede realizar tantas veces como se desee dentro del cuerpo del programa (al igual que una función), y consistirá en el nombre del procedimiento, seguido entre paréntesis de los valores que se den a los parámetros formales incluidos en la declaración, separados por comas. El número de valores introducidos en el procedimiento deben ser igual al número de parámetros establecidos en la declaración del procedimiento, y cada uno de ellos debe ser de igual tipo que el parámetro al que sustituya. Hay que tener especial cuidado en la llamada a un procedimiento, pues variar el orden de los valores de los parámetros en una llamada a procedimiento implica la incorrección lógica del programa. Por esta razón es aconsejable incluir comentarios tras la cabecera de un procedimiento, indicando el tipo de parámetro que se espera recibir junto con la función que realizan.

Cuando se produce una llamada a un procedimiento, se sustituyen los parámetros formales (aquellos que fueron definidos en la cabecera del procedimiento), por los parámetros actuales o argumentos, que están presentes en la llamada. El control del programa lo tendrá el procedimiento, es decir, se ejecutará el cuerpo del mismo, y cuando éste termine pasará el control a la siguiente instrucción del programa principal.

La división de un programa en módulos facilita la programación

```
funcion factorial (x: entero): entero
  acumulador: entero
  contador: entero
inicio
  acumulador = 1
  desde contador = 1 hasta x
            acumulador = acumulador * contador
  factorial = acumulador
fin factorial

procedimiento intro_datos (  num_pos: entero)
inicio
  repetir
     borrar_pantalla
     Escribir ('Introducir un número positivo ')
     Leer ( num_pos )
  hasta  num_pos > 0
fin intro_datos


programa  muestra
   numero: entero
   resultado: entero
inicio
   intro_datos (numero)
   resultado = factorial ( numero )
   Escribir ('El factorial de  ', numero, ' es ',
resultado)
fin muestra
```

Cuadro 1.

```
unit <nombre_de_unidad>;
interface
uses <lista_de unidades>;

    {DECLARACIONES DE DOMINIO PUBLICO}

implementation

    {DECLARACIONES DE DOMINIO PRIVADO}
    {CUERPO DE PROCEDIMIENTOS Y FUNCIONES}

begin

    {CODIGO DE INICIALIZACION}

end.
```

Cuadro 2.

Como se ha visto hasta ahora, los procedimientos y funciones tienen fines similares, sin embargo existen una serie de diferencias entre ellos:

1.- Una función devuelve un único valor, mientras que los procedimientos no devuelven valores de manera explícita, lo hacen a través de sus parámetros.

2.- El nombre de procedimiento no está asociado a ningún resultado, en cambio el de una función si.

3.- La llamada a una función siempre se realiza dentro de una expresión, mientras que la llamada a un procedimiento se realiza en una línea de programa, como si se tratase de una instrucción simple.

ESTRUCTURA GENERAL DE UN PROGRAMA

Hasta ahora, un programa se componía de una cabecera de programa, junto con un conjunto de declaraciones y el cuerpo del programa (conjunto de instrucciones). La introducción de funciones y procedimientos, modifican la estructura del programa. Si el programa es "sencillo", se puede incluir en el mismo fichero fuente todos los procedimientos y funciones creadas por el usuario para la realización del programa. Por ejemplo, en Pascal, los procedimientos y funciones definidos por el usuario se sitúan entre la declaración de variables y el cuerpo del programa. En BASIC y C, las funciones y procedimientos (en el caso de C, no hay procedimientos), se sitúan tras el cuerpo del programa principal.

La codificación de un programa se facilita dividiéndolo en subprogramas

En caso de tener que construir un programa extenso, existe la posibilidad de recurrir a la compilación separada. Ésta consiste en dividir el programa en varios módulos situados en diferentes ficheros fuente, compilar estos módulos independientemente del programa principal y más tarde construir el programa principal, utilizando las funciones y procedimientos de los módulos, tras enlazar todos los módulos, se obtiene el fichero ejecutable que se corresponde con el programa final.

Normalmente, los entornos integrados disponen de varias herramientas que facilitan la compilación separada (normalmente las opciones dentro de la opción de compilación, *make* y *build*).

En este artículo, comentaremos la estructura de un módulo en Turbo Pascal, denominado unidad (*unit*), que por sus características es muy adecuado para el diseño descendente. Una unidad es una librería de declaraciones de constantes, tipos de datos, variables, procedimientos y funciones, que podemos usar en cualquier programa que los necesite, y que permite dividir el programa principal en elementos más sencillos, además de poder ser compilado independientemente de él. Normalmente, en una unidad se incluyen declaraciones de objetos que están relacionados entre sí.

La sintaxis de una unidad está diseñada de tal forma que ofrezcan una serie de servicios a través de procedimientos y funciones, pero ocultando su implementación, de esta forma sabemos que si en una unidad se encuentra disponible la función *factorial*, podemos utilizarla pero no se nos da a conocer como está construida, qué variables internas utiliza etc., sólo sabemos que si damos un valor entero positivo a esta función, se devolverá el valor de su factorial. Este hecho permite a los programadores utilizar funciones y procedimientos ya implementados sin conocer como están construidos, solamente debe conocer el interfaz de los subprogramas y lo que realizan, no cómo lo realizan.

La estructura de una unidad se muestra en el cuadro 2.

La cabecera de una unidad comienza con un nombre de unidad, acompañado de la palabra reservada *unit*, de manera similar a como ocurre en un programa. A continuación viene la palabra reservada *INTERFACE*, en esta sección de la unidad se puede incluir una cláusula uses, indicando a continuación las unidades que utiliza la unidad que se está construyendo (aquí se puede observar como Turbo Pascal permite de una forma muy sencilla el diseño descendente), posteriormente se incluyen las definiciones de constantes, variables y tipos propios de la unidad actual y a continuación las cabeceras de los procedimientos y funciones que ofrecerá esta unidad a

```
PROGRAMA PRINCIPAL
  variables X Y Z

  Procedimiento_1
    variables X R S
        |  No es visible la variable X del programa
        |  Principal, cualquier uso de la variable
        |  X se referirá a la variable del procedimiento_1
  Fin del Procedimiento_1

  Procedimiento_2
    variables Y Z T
        |  No son visibles las variables Y y Z del
        |  programa principal. No son visibles las
        |  variables del procedimiento_1.
  Fin del Procedimiento_2

Inicio del Programa principal
        |  Son visibles las variables definidas en el programa
        |  principal. No son visibles las variables definidas en
        |  los procedimientos.
Fin del Programa principal
```

Cuadro 3.

otras unidades, a todos estos elementos se les denomina visibles, pues son utilizables por otras unidades. Tras finalizar las declaraciones visibles, viene la palabra reservada *IMPLEMENTATION*, tras la cual vienen las declaraciones no visibles y la implementación de la funciones y procedimientos cuyas cabeceras aparecían en la sección *INTERFACE*. Además se pueden incluir otros procedimientos y funciones que pueden utilizarse para implementar a las funciones y procedimientos visibles. Todos estos elementos, no serán utilizables por los programas que hagan uso de la unidad, solamente se podrá hacer uso de los elementos declarados en la sección *INTERFACE*. Por último, al final de la unidad, y entre un BEGIN y un END, se podrán incluir instrucciones para inicializar las variables declaradas en la unidad.

Desgraciadamente, la mayoría de los lenguajes no disponen de estructuras similares a las *units* de Turbo Pascal, utilizan simples librerías de funciones y procedimientos.

ÁMBITO DE UNA DEFINICIÓN

Con la introducción de funciones y procedimientos se amplían los lugares donde definir variables, así es posible declararlas tanto en el programa principal, como en las funciones y procedimientos. Surge así el concepto de ámbito de una definición, refiriéndose al lugar del programa donde se podrán utilizar las variables declaradas en diferentes partes del programa.

Variables locales son aquellas que han sido declaradas dentro de un procedimiento o una función

Se define variable global, como aquella que se ha declarado en el programa principal, estas variables pueden ser utilizadas en cualquier lugar del programa, incluso por los procedimientos y funciones declarados en él. Llamaremos variables locales, a aquellas que han sido definidas en una función o procedimiento. Estas variables

BUSCAMOS A LOS MEJORES

- Programadores en C/C++
- Grafistas, diseñadores, dibujantes
- Expertos en lenguaje ensamblador
- Programadores 3D
- Expertos en Sonido
- Músicos con nociones de MOD, MIDI
- Programadores de juegos
- Animadores gráficos
- Infografistas con experiencia en 3D Studio, Photoshop, Deluxe Paint Animation
- Programadores con dominio de lenguajes multimedia: Authorware, Toolbook, Visual Basic, Macromedia Director, etc
- Expertos en comunicaciones

PARA DESARROLLAR

SOFTWARE MULTIMEDIA:

Presentaciones, libros interactivos, programas educativos, sistemas de comunicaciones, centros servidores de datos, videojuegos y mucho más...

SI ERES programador, músico o infografista y tienes ideas o proyectos en estudio de desarrollo ponte ya en contacto con nosotros.

TE OFRECEMOS las mejores condiciones y apoyo para producir tus programas y venderlos en el mercado nacional y extranjero. Formación en nuevas tecnologías y la mejor biblioteca de rutinas gráficas y sonido para desarrolladores.

Aportamos gráficos, rutinas o música, según las necesidades, para complementar cada proyecto.

Si estás interesado en unirte a una de las empresas más punteras en alta tecnología y desarrollo de software, no dejes pasar esta oportunidad. Envíanos carta con tus datos personales (curriculum vitae, con una muestra de tus anteriores trabajos) y un teléfono de contacto a:

DDM DIGITAL DREAMS MULTIMEDIA

DIGITAL DREAMS MULTIMEDIA
Ref. Programadores
C/ Vicente Muzas 15, 1° D - 28043 MADRID
Tf.: (91) 519.23.53 Fax (91) 413.55.77
BBS: (91) 519.75.75 Internet: ddm@servicom.es

tendrán su ámbito solamente en el procedimiento o función donde han sido definidas.

Existen varios puntos a aclarar. Si se ha utilizado el mismo identificador para dos declaraciones, una en el programa principal y otra en una función o procedimiento, el ámbito de la variable local seguirá siendo la función o procedimiento donde fue definida, pero el ámbito de la variable global será todo el programa, excepto el ámbito de la variable local, es decir, la variable local "oculta" a la variable global en el lugar donde la variable local ha sido definida. Este hecho se complica en lenguajes que permiten el anidamiento en la definición de procedimientos y funciones, es decir, dentro de un procedimiento o función se permite la declaración de funciones y procedimientos, siendo las reglas de ámbito entre el procedimiento o función más interno, con respecto al procedimiento o función más externo, el mismo que entre el procedimiento o función y el programa principal, es más, los procedimientos y funciones definidas dentro de una función o procedimiento, no serán utilizables por el resto del programa, pudiendo ser utilizados solamente dentro del cuerpo del procedimiento o función en el que fue definido.

La metodología "Divide y Vencerás" ayuda a realizar programas mucho más estructurados

Una buena costumbre a la hora de programar, es evitar utilizar variables globales dentro de los procedimientos y funciones definidos dentro de un programa, la comunicación entre los programas y subprogramas se deberá realizar a través del paso de parámetros, esto permite dar mayor independencia a la codificación del programa y sus subprogramas(cuadro 3).

Existen también en numerosos lenguajes de programación, la posibilidad de modificar el comportamiento de una variable en cuanto a su ámbito (por ejemplo, en C se puede modificar el ámbito de una variable definiéndola como extern).

PASO DE PARÁMETROS

Los parámetros definidos en la cabecera de una función o procedimiento permiten a éstos comunicarse con el programa. Cuando se realiza una llamada a un subprograma, ésta debe exhibir tantos argumentos (valores que tomarán los parámetros del subprograma en la llamada), como parámetros se hayan definido en la cabecera del subprograma. El orden de asociación entre los parámetros y argumentos será posicional, es decir, el primer argumento se asociará al primer parámetro de la definición del subprograma y así sucesivamente con el resto de los argumentos y parámetros.

Existen diferentes métodos en el paso de parámetros, dependiendo de la utilización que deseemos hacer de éstos. Si lo que se desea es dar a conocer un valor a la función o procedimiento, pero sin modificarlo, entonces realizaremos el paso de parámetro por valor, si por el contrario lo que se desea es modificar el valor de una variable externa al procedimiento utilizaremos el paso por variable (también conocido como paso por referencia).

En general, los subprogramas pueden ser funciones o procedimientos

El paso de parámetros por valor se realiza evaluando la expresión del argumento pasado como parámetro, lo que realmente comunica el procedimiento o función es el resultado de la expresión, es decir, su valor. Por tanto, el argumento correspondiente a un parámetro puede ser tanto una variable como una expresión, cuyo resultado debe ser de igual tipo que el definido en la cabecera de un procedimiento o función para el parámetro correspondiente.

En el paso de parámetro por variable, lo que realmente se comunica al procedimiento es la dirección de memoria en la que se puede encontrar la variable pasada como argumento. En este tipo de paso de parámetro sólo se permite, como es lógico, que al parámetro se le asocien variables del mismo tipo. Cualquier modificación que se realice dentro del procedimiento con el parámetro, se realizará sobre el argumento con el que se llamó al procedimiento.

Por tanto, si lo que se desea es modificar los argumentos mediante un procedimiento, éstos deben pasarse mediante un parámetro por variable, si no deseamos que el procedimiento modifique el argumento, éste debe ser transmitido por valor.

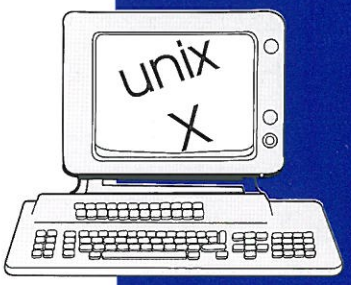
Típicamente, las funciones en los lenguajes de programación sólo pueden tener parámetros por valor, transmitiendo el resultado de la función a través de una instrucción específica (return en C o asignando un valor al nombre de la función en Pascal). Por el contrario, los procedimientos pueden manejar los dos tipos de paso de parámetros, siendo capaces de devolver varios resultados gracias a la utilización de parámetros por variable.

En Pascal, especificamos parámetros por variable, antecediendo a éstos la cláusula var, en la cabecera de definición del procedimiento. Aquellos parámetros que no estén precedidos por esta cláusula serán pasados por valor.

En C, solamente existen funciones que, como se comentó con anterioridad, sólo admiten el paso de parámetros por valor. La solución para poder modificar un parámetro, consiste en pasar como parámetro un puntero, que es sencillamente un valor que contiene la dirección de memoria donde se encuentra la variable que se desea modificar, haciendo posible la modificación de argumentos, aunque no explícitamente. ■

EL SISTEMA DE VENTANAS X WINDOW

Fernando J. Echevarrieta



En el presente artículo se presenta la interfaz gráfica por excelencia del mundo UNIX. A través del mismo, se repasarán tres aspectos del entorno: 1. Descripción del sistema y explicación de cómo funciona, 2. Notas de instalación y resolución de problemas y 3. Cómo personalizarlo según los gustos de cada usuario. Con ello, el lector deberá ser capaz de trabajar en un entorno agradable y personalizado.

X WINDOW es la interfaz gráfica estándar para máquinas UNIX. A través de esta interfaz, es posible disponer de varios terminales virtuales en la misma pantalla, e incluso generar una sesión de login distinta en cada uno de ellos. El manejo se realiza a través de un dispositivo puntero (ratón, trackpoint, trackball) y, aunque su uso no es necesario, pensar en X WINDOW sin este dispositivo es como pensar en MS WINDOWS sin el mismo. Sin embargo, la utilización del sistema de ventanas X, aporta mucho más que la mera posibilidad de disponer de varios terminales virtuales en una única pantalla. Así, en toda distribución (incluida la de Linux) pueden encontrarse aplicaciones como *xterm* (emulador de terminal), *twm* (*window manager*), *xconsole* (programa redirector de consola), *xman* (manual basado en menús), *xclock* (reloj gráfico), *xload* (monitor de carga del sistema) y muchas otras. Además de estas aplicaciones básicas existen miles de otros programas creados para X, desde clientes de WWW hasta complejos programas de CAD. Como ante todo producto, surge la duda de si merece la pena su uso y disfrute (que en este caso supone una ardua labor de instalación y configuración) y, como en todos los casos, depende de lo que uno espere de su sistema. Lo cierto es que en el caso de un PC soportando Linux (u otro UNIX para familia x86), la utilización del sistema será equivalente a disponer de una estación gráfica. En cualquier caso es importante aclarar que X WINDOW es únicamente una interfaz gráfica (no como MS WINDOWS): todo el soporte multiproceso lo proporciona el propio sistema operativo.

SERVIDORES Y CLIENTES

El sistema de ventanas X WINDOW incluye, además, una filosofía que proporciona un soporte para *displays* remotos con una concepción de red en la que existen dos tipos de servidores: 1. orientados a cálculo y 2. servidores gráficos o *workstations*. Esta filosofía se basa en una arquitectura *cliente-servidor* como la que se muestra en la figura 1. Según esta arquitectura, debe existir un servidor de X en la máquina en cuyo *display* aparecerán las ventanas mientras que los clientes, es decir, las aplicaciones que serán mostradas, podrán residir en la misma máquina y/o en máquinas remotas distintas. Asimismo, un cliente podrá verse representado por una o múltiples ventanas. El funcionamiento del sistema está orientado a eventos, y basado en ejecución de procedimientos remotos de tal forma que una aplicación cliente "dibuja" en un *display*, local o no, enviando mensajes al servidor de la máquina destino. En estos mensajes se indican las instrucciones necesarias para generar el "dibujo" que debe ir realizando.

EL WINDOW MANAGER

El *window manager* o gestor de ventanas, es la aplicación más importante que utilizará los servicios prestados por el servidor X. Será el responsable de lo que en terminología anglosajona se denominaría el *look & feel* del sistema. Mientras el servidor X sólo "sabe" dibujar, será el *window manager* el programa que gestionará y adornará todas las ventanas. Determinará su posición y aspecto, y presentará una interfaz de usuario unificada proporcionando métodos (botones, barras de menú, menús *pop-up*) para su dimensionamiento, ico-

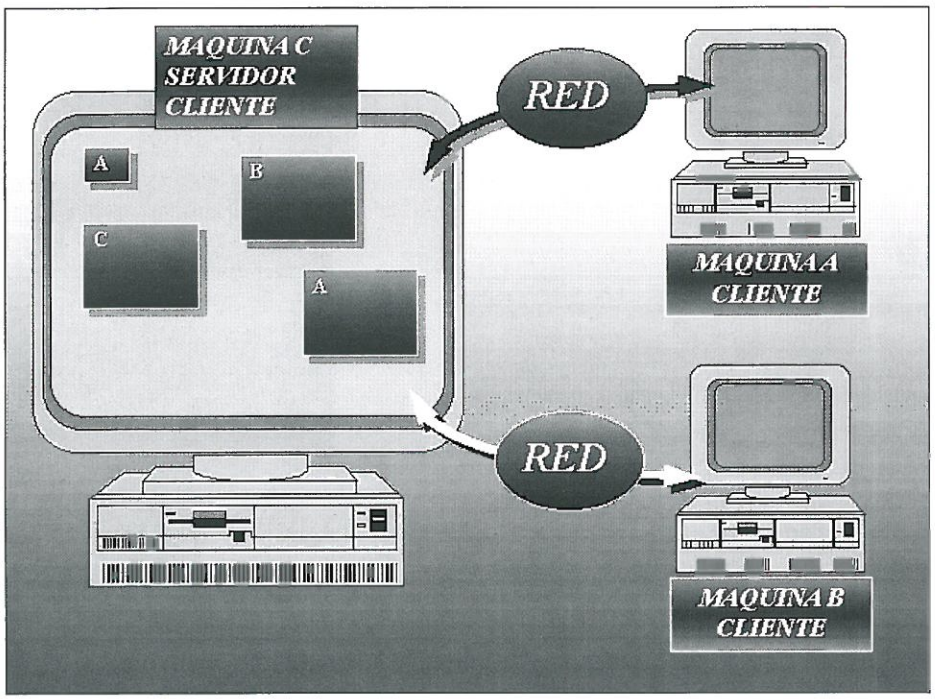


Figura 1. Arquitectura Cliente-Servidor de X-WINDOW.

nificación, desplazamiento, etc. Sin embargo, no hay que olvidar que se trata de un cliente más, por lo que será totalmente intercambiable. Está es la razón por la que cada persona dispone de una interfaz diferente. Para concretar, se podría decir que un *window manager* sería lo que a las MS WINDOWS les colocaría una barra de título superior, botones de maximizar, minimizar e iconificar a la derecha, barra de menú bajo el título, etc. La distribución base de X incluye uno llamado *twm*, que por esta razón ha sido uno de los más extendidos, cuya apariencia se puede observar en la foto 1. Sin embargo, con posterioridad han ido surgiendo otros muchos. En el mundo Linux, se considera estándar el *fvwm*, un *virtual desktop* o escritorio virtual que permite utilizar un área de trabajo de superficie mucho mayor a la de la pantalla.

LAS "RELEASES" Y SUS REQUERIMIENTOS

El desarrollo original del sistema se realizó en el MIT (Instituto Tecnológico de Massachusetts) y pertenecía a la modalidad de libre distribución. Posteriormente se han realizado distintas versiones, siendo la más extendida en el mundo PC la distribución gratuita *XFree86*, que forma parte de las distribuciones de Linux que acompañaban a los números 2, 4 y 6 de Sólo Programadores. En los dos primeros casos se entregó la versión XFree86-2.0 que correspondía al sistema X11R5 (X11 Release 5), y en el último, la versión XFree86-3.1 que corresponde a X11R6. La última versión hasta el momento de la redacción de este artículo es XFree86-3.1.1, que resuelve algunos problemas en algunos de los servidores de su predecesora. La *release* 6 es la última implementación del sistema, por lo que en un futuro cercano la R5 habrá caído en desuso en el mundo UNIX. Sin embargo, es interesante considerar algunos puntos antes de decidirse por una u otra en una plataforma PC.

Los requerimientos mínimos para utilizar el sistema con Linux son los mismos que para el sistema base: un procesador 386DX y 4 Mb de RAM (y, por supuesto, una partición de *swap*). X WINDOW consume mucha memoria, así que ésta será un factor determinante, llegando a ser mucho más rápido un 386 con 8Mb, que un 486 con sólo 4Mb.

XFree86 proporciona servidores para multitud de tarjetas gráficas, desde VGA estándar hasta SVGA, e incluso servidores específicos para tarjetas aceleradoras. En este aspecto, la oferta de X11R6 es mucho más completa, tanto en calidad como en cantidad y, además proporciona una compatibilidad hacia atrás.

CONFIGURACIÓN DEL SERVIDOR XFree86

La configuración del servidor es una labor de administración del sistema, por lo que habrá que actuar como *root*. No ocurre así con la personalización del entorno (lanzamiento de clientes y configuración del aspecto en pantalla), que es particular para cada usuario, como se verá más adelante. Es por ello que los siguientes apartados relativos al servidor harán referencia a XFree86 para la plataforma PC. Aunque se tomará como ejemplo la distribución de Linux, hay que aclarar que los usuarios de otros sistemas UNIX, como FreeBSD, BSDI, CurrentBSD o NetBSD, también pueden disfrutar de XFree86.

Tanto si se ha optado por instalar X11R5 como X11R6 a partir de las distribuciones de Sólo Programadores, se deberá seleccionar la serie de discos X. Para cualquier duda, puede consultarse el libro "LINUX, UNIX para todos" que acompañaba al número 9 de la revista. Con X11R6, la adaptación al hardware se puede hacer de forma automática a través del programa:

/usr/X386/lib/ConfigXF86/xf86config-XFree86-3.1

Únicamente en caso de que esta instalación no se realice normalmente o se haya optado por instalar X11R5 habrá que realizar la configuración a mano, para lo que se indican algunas notas en el siguiente apartado.

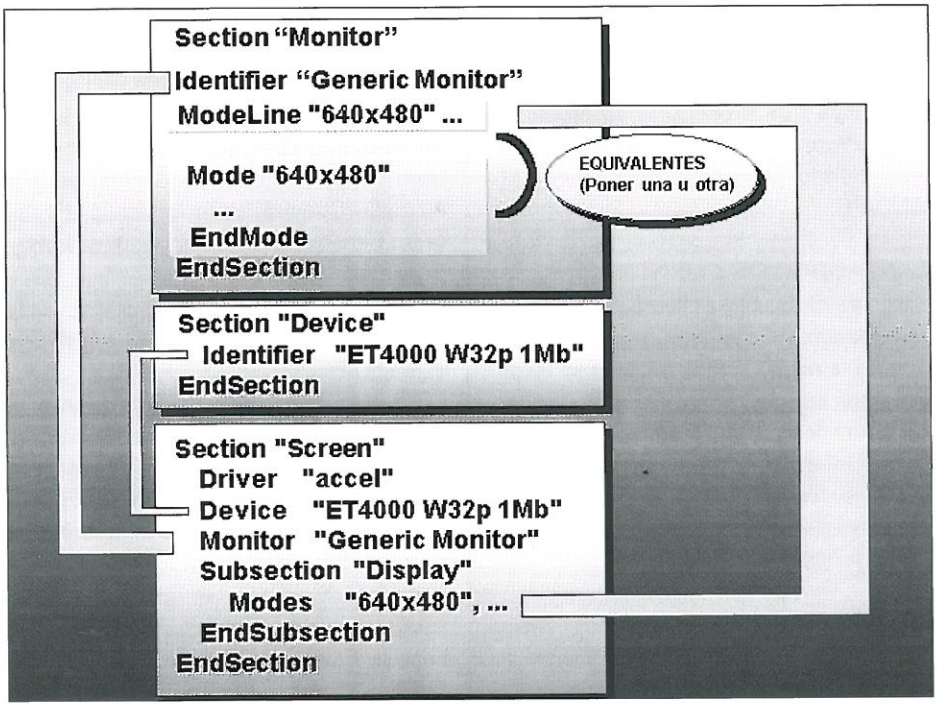
CONFIGURACIÓN AVANZADA

El fichero responsable de la configuración del servidor X (el equivalente al SYSTEM.INI en MS WINDOWS) ha variado notablemente de una versión a otra. Así, hasta X11R5 inclusive, recibe el nombre de *Xconfig* y, a partir de X11R6, el de *XF86Config*. En ambos casos, se encuentra en el directorio */usr/X386/lib/X11*. Realmente, cada distribución se instala en lugares diferentes, pero se establecen automáticamente enlaces simbólicos para que los nombres de los directorios siempre coincidan. Básicamente se puede decir que la diferencia entre uno y otro fichero es que a XF86Config se le ha dotado de estructura, algo de lo que carecía Xconfig. Con ello se permite que pue-

dan coexistir en el mismo fichero distintos servidores, con distintas tarjetas, y distintos monitores, todos a la vez y con una configuración distinta para cada uno. El problema es que, por ello, el fichero resulta más complejo y, si no se conoce la estructura, puede hacer también más compleja su configuración. Para aquellos usuarios que deseen, o se vean obligados a modificarlo a mano, la figura 2 muestra las relaciones que deben existir entre secciones interrelacionadas. Otras, como la configuración del ratón, no deberían crear problemas y se pueden realizar automáticamente. En cualquier caso, leyendo el fichero y comparándolo con la ilustración debería identificarse claramente la estructura. Todas las secciones comienzan por la palabra clave *Section* y terminan con *EndSection*. La palabra *Identifier* se utiliza para dar nombre a una sección concreta y se puede utilizar cualquier cadena de texto, pero será este nombre el que servirá para ligar las secciones como se puede apreciar en la figura. En el mismo fichero se pueden crear varios grupos de secciones "Monitor" (descripción del monitor), "Device" (tarjeta de vídeo) y "Screen" (servidor y resolución) siempre que se respeten las relaciones de la figura. El servidor que haya sido instalado buscará en las secciones "Screen" hasta encontrar el "driver" que le corresponde y, a partir de ahí, sabrá con qué secciones "Monitor" y "Device" está ligado. El fichero Xconfig de X11R5 es mucho más simple. Si se desea más información, en el directorio */usr/X386/lib/X11/doc* se pueden encontrar ficheros README.* relativos a distintos tipos de tarjetas y el fichero *VideoModes.doc* de 51 kb que explica con absoluto detalle el funcionamiento de un monitor de rayos catódicos, y como ajustar ciertos valores críticos de forma manual.

```
PACKAGE NAME:     vgaset
COMPRESSED PACKAGE SIZE:    14 K
UNCOMPRESSED PACKAGE SIZE:     40 K
PACKAGE LOCATION: disk xap1

CONTROLES:
l  - Reduce  el margen izquierdo
L  - Aumenta el margen izquierdo
r  - Reduce  el margen derecho
R  - Aumenta el margen derecho
h  - Reduce  el ancho de pulso del sincronismo
     horizontal
H  - Aumenta el ancho de pulso del sincronismo
     horizontal
t  - Reduce  el margen superior
T  - Aumenta el margen superior
b  - Reduce  el margen inferior
B  - Aumenta el margen inferior
v  - Reduce  el ancho de pulso del sincronismo
     vertical
V  - Aumenta el ancho de pulso del sincronismo
     vertical
x  - Reduce  la resolucion horizontal
X  - Aumenta la resolucion horizontal
y  - Reduce  la resolucion vertical
Y  - Aumenta la resolucion vertical
?  - Vuelta a los valores iniciales
CR - Vuelve a mostrar los ajustes
q  - Salir
```



Relación entre las secciones de XF86Config

UN AJUSTE FINO MUY SIMPLE

Bien se haya realizado la configuración automáticamente, bien se haya realizado a mano, es posible que la pantalla aparezca desplazada vertical u horizontalmente, demasiado pequeña o demasiado grande. Para realizar un ajuste fino, se puede ejecutar el programa *vgaset* presente en la serie de discos XAP, desde un xterm. Una vez se visualice la pantalla del modo deseado, bastará copiar los valores de los *timings* del monitor que aparecen en pantalla a la sección "ModeLine" o "Mode" (ver figura 2) del fichero XF86Config si se ha instalado X11R6, o a la sección "ModeDB" del fichero Xconfig, si se ha instalado X11R5, para que el sistema arranque siempre ajustado. El listado 1 presenta los controles de ajuste que proporciona el programa.

PERSONALIZACIÓN Y ARRANQUE

Como adivinarán la mayoría de los lectores, la personalización del sistema (entendiendo por personalización la configuración a nivel de cada usuario, no la global a nivel de administración) se llevará a cabo simplemente editando un fichero de texto presente en el directorio *home*. El fichero en cuestión recibe el nombre de *.xinitrc*. El arranque de las X deberá realizarse mediante la instrucción *startx*, que en realidad es un *script* (programa *batch*) que se encargará de lanzar otro programa llamado *xinit*, pasándole los argumentos necesarios para el correcto lanzamiento de servidor y clientes X. Al arrancar de esta forma, el sistema buscará el fichero *$HOME/.xinitrc* ($HOME es una variable *shell* que indica el directorio *home* del propietario de la shell) y en caso de no encontrarlo, recurrirá al */usr/X386/lib/X11/xinit/xinitrc*. El fichero .xinitrc será interpretado como un *script* que lanzará los clientes deseados del modo adecuado para un correcto funcionamiento. "Modo adecuado" significa, que no deberán tomar control del terminal ya que no se ejecutaría un cliente hasta que hubiera terminado el anterior. Por tanto, deberán ejecutarse

en paralelo en *background*, y no de modo secuencial salvo que se sepa que el cliente va a morir rápidamente y ceder el control. El último cliente sí deberá ejecutarse en *foreground* de tal forma que el *script* sólo terminará una vez que éste haya finalizado. Es por ello que este último cliente suele ser el propio *window manager*. De esta forma, para salir de una sesión X bastará elegir la opción "Exit" del mismo.

La utilización de xinit directamente y no a través de *startx*, se reserva a usuarios experimentados. El servidor X arranca y lanza la aplicación por defecto *xterm -geometry +1+1 -n login -display :0*. Como se puede apreciar, no existe *window manager*.

En el listado 2 se muestra un típico .xinitrc. En el se puede apreciar que comienza utilizando el comando *xrdb -load* para cargar el fichero *.Xdefaults* de recursos para las X. En este fichero

LISTADO 2

```
xrdb -load $HOME/.Xresources
xsetroot -solid SteelBlue &
xclock -g 100x100+520+10 -update 60 &
xterm -ls -fn 6x13 -g 80x24+10+70 &
xeyes -g 37x33+400+10 &
color_xterm -ls -fn 6x10 -g 80x24+140+200 &
fvwm
```

xinitrc.

figuran los valores por defecto de parámetros como colores de fondo y primer plano, tipos de fuente de letra, tamaño de barras de desplazamiento, etc. En cualquier caso, existen unas opciones definidas por defecto que en Linux se encuentran en */usr/X386/lib/X11/xdm/Xresources* y en */usr/X386/lib/X11/app-defaults*.

Continuando con el listado, aparece otro nuevo comando: *xsetroot* que se utiliza para configurar el fondo de las X. En este caso, mostrará un fondo de color azul acero sólido. Aunque se ha ejecutado en *background*, el programa devuelve el control nada más ajustar el fondo, por lo que no habría sido necesario. La base de datos con los nombres de colores se encuentra en */usr/X386/lib/X11/rgb.txt*.

Tras configurar el fondo, el fichero .xinit de ejemplo lanza algunos clientes X en paralelo. Todos ellos, y muchos otros, se encuentran en Linux en */usr/X386/bin*. Por último, como se recomendaba, se lanza el gestor de ventanas, esta vez, en *foreground*.

El sistema X, también ofrece la posibilidad de personalizar el teclado. Para utilizar el teclado en castellano, se podría haber ejecutado *xmodmap /etc/modmap.es*.

UN PROBLEMA DE SEGURIDAD

Es necesario conocer, que al arrancar el sistema conmuta automáticamente a la consola numero 7, por lo que la sesión con la *shell* inicial continuará en la consola n desde la que se lanzó el servidor. Esto puede plantear un problema de seguridad ya que, aunque se bloquee la consola que se encuentra en modo gráfico (con *xlock*, por ejemplo), durante alguna ausencia (*cofee break*, pipi room...), alguién podría conmutar a la consola original pulsando CTRL+ALT+Fn donde encontraría la *shell* madre abierta y a su disposición. Para usarla, podría abortar el servidor (CONTROL-C) o, simplemente, pararlo (CONTROL-Z), y dispondría de la línea de comando como si fuera el legítimo propietario. Es por ello que conviene, en caso de trabajar únicamente en modo gráfico o de ausentarse, que el propietario realice esta acción, parando el servidor, y enviándolo a *background* (con *bg*, ver el capítulo correspondiente del curso) y, posteriormente, salir de la sesión, con lo que volverá a aparecer el mensaje de *login*. El servidor no morirá ya que proseguirá su ejecución en *background* y podrá volverse a él accediendo a su consola mediante ALT+F6.

LANZANDO LOS CLIENTES

En principio, todo cliente X puede ser lanzado sin indicar ningún parámetro ya que en la mayoría de los casos se tomarán unos valores por defecto. Sin embargo, entre los parámetros comunes a las aplicaciones X se encuentran:

-g o -geometry anchoxalto+offset(horizontal)+offset(vertical)

donde ancho y alto son números enteros que expresan un tamaño en *pixels* o en caracteres según se trate de una ventana gráfica o de texto. Los *offset* son distancias en *pixels* desde el origen de coordenadas (si son positivos), situado en la esquina superior izquierda, o hasta la esquina inferior izquierda (si son negativos).

-display host:display.pantalla

donde *host* indica la máquina donde se encuentra el servidor X y para *display.pantalla*, en principio se tomará siempre *0.0*. Hay que destacar que para

LISTADO 3

```
xrdb -load $HOME/.Xresources
xsetroot -solid SteelBlue &
xclock -g 100x100+520+10 -update 60 &
sleep 1
xterm -ls -fn 6x13 -g 80x24+10+70 &
sleep 2
xeyes -g 37x33+400+10 &
sleep 1
color_xterm -ls -fn 6x10 -g 80x24+140+200 &
sleep 2
fvwm
```

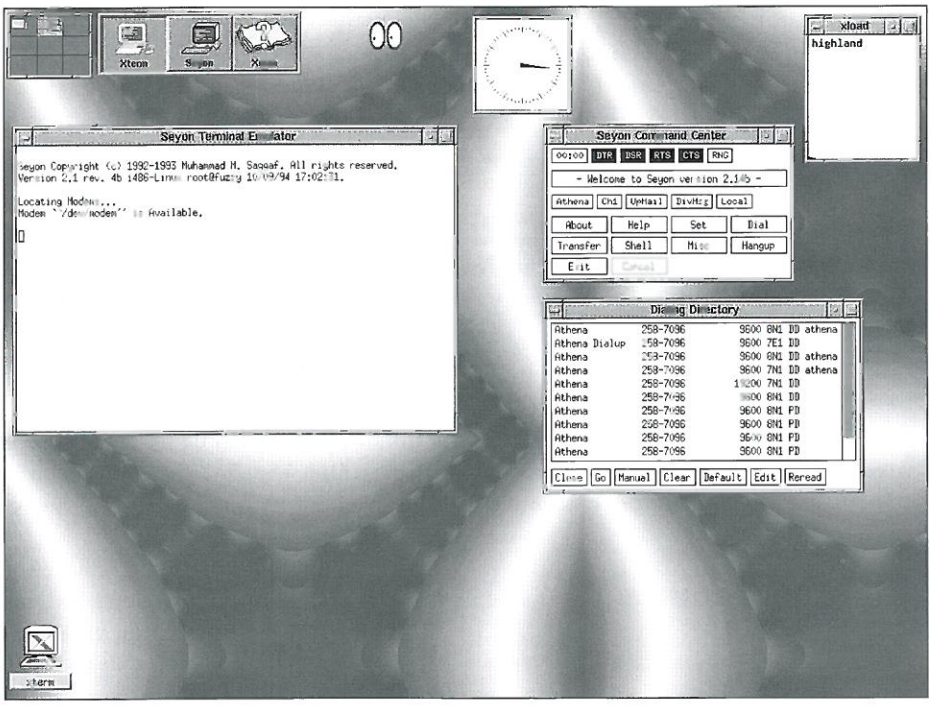
xinitrc para equipos lentos.

poder utilizar el servidor de una máquina, ésta debe permitir el acceso al ordenador remoto donde se ejecuta el cliente. Para gestionar estos accesos se dispone del comando *xhost [+-][maquina...]*, que permite (+) o deniega (-) el acceso a las máquinas especificadas o a todas si no se especifican.

En sistemas lentos, conviene lanzar los procesos en *background*, como es imperativo, pero de una manera secuencial, permitiendo que un proceso se cargue completamente antes de tratar de iniciar el siguiente (nótese que se trata de lanzamiento secuencial pero la ejecución se realiza en paralelo).El listado 3 ilustra uno de los posibles lanzamientos.

PERSONALIZACIÓN DEL *WINDOW MANAGER*

Como se comentaba anteriormente, el *window manager* es el cliente que se encarga de configurar la presentación de las aplicaciones en pantalla. Será, por tanto, este programa el que habrá que configurar según los gustos de cada uno. Como ya se ha expuesto, es esta característica de ser un cliente más lo que hace que existan multitud de *window managers*. Como de costumbre, el fichero a modificar por el usuario



Aplicación de comunicaciones *seyon* con el *window manager* configurado según el texto del artículo.

se encuentra en el directorio *home* del mismo y recibe el nombre de *.fvwmrc*. En caso de no tenerlo, se estará utilizando el fichero por defecto para todos los usuarios: */usr/X386/lib/X11/fvwm/system.fvwmrc*. Para personalizarlo, en este caso, bastará copiarlo al directorio *home*, y realizar los cambios deseados. Se trata de un fichero muy extenso, sin embargo, es muy sencillo de configurar. Afortunadamente, dispone ya de muchas configuraciones que aparecen como comentarios, y que permiten aprender rápidamente sobre él. Como primer contacto, se puede tratar de añadir, cambiar o eliminar aplicaciones del menú. Estas figuran en líneas autoexplicativas, como:

Exec "Seyon" exec seyon -modem /dev/modem &

que hará que la aplicación de comunicaciones seyon que se puede observar en la foto 5 figure en el menú. Como se puede apreciar, es posible indicar opciones y parámetros del mismo modo que se haría desde la *shell*. Una vez más, habrá que lanzar los clientes en *background*.

Otra opción configurable muy común son los estilos de las aplicaciones, mediante los cuales se determina si las ventanas de cada aplicación concreta deben estar siempre en primer plano, llevar título, bordes, su grosor, si deben aparecer en todas las páginas del *virtual desktop* siguiendo los movimientos del usuario (ser pegajosas), etc. En fvwm se realizan mediante líneas como:

Style "GoodStuff" NoTitle, NoHandles, Sticky, WindowListSkip,BorderWidth 5
Style "Fvwm Pager" StaysOnTop
Style "xeyes" NoTitle, Sticky, WindowListSkip
*Style "*lock" NoTitle, NoHandles, Sticky, WindowListSkip*
Style "xcalc" Icon xcalc.xpm

donde, en el caso de "*lock", se puede apreciar que se pueden utilizar comodines para hacer referencia a varias aplicaciones.

Respecto al manejo de iconos, .fvwmrc presenta la instrucción IconBox, que define una geometría para el lugar donde se situarán los iconos de las aplicaciones que queden minimizadas. Así, por ejemplo, para la resolución 640x480:

IconBox 10 380 630 470

definirá una zona a pie de pantalla.

Especialmente interesante en equipos lentos, es la orden: *OpaqueMove*, que determina el tamaño de ventana, expresado en tanto por ciento de la pantalla total, a partir de la cual en lugar de arrastrar toda la ventana con su contenido se arrastrará únicamente su silueta, y sólo se redibujará cuando se haya elegido la posición determinada y se suelte el botón del ratón. Así, *OpaqueMove 50* indica que las ventanas que ocupen menos del 50% de la pantalla se arrastrarán con su contenido, y el resto prestará únicamente su silueta al arrastre.

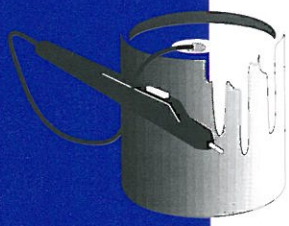
Fvwm también permite indicar ciertas acciones a tomar al arrancar, como si se tratara de un autoexec.bat propio. Ésto se realiza a través de la función "InitFunction". Un posible ejemplo sería:

Function "InitFunction"
Module "I" GoodStuff
EndFunction

que lanza la aplicación *GoodStuff*, un cliente del *window manager*, una de las innovaciones de fvwm. *GoodStuff* aparece como una botonera en pantalla que también se puede observar en la foto 5, donde tiene definidos 3 botones: *xterm*, *seyon* y *xman*. Para realizarla hubo que incluir las siguientes líneas en el fichero .fvwmrc:

**GoodStuffGeometry +105+5*
**GoodStuff Xterm rterm.xpm Exec "Xterm" xterm &*
**GoodStuff Seyon term.xpm Exec "Seyon" seyon -modems /dev/modem &*
**GoodStuff Xman /usr/local/lib/pixmaps/xman.xpm Exec "Xman" xman &*

donde los ficheros con extensión *.xpm*, son los iconos elegidos. Como se puede apreciar, existen una infinidad de posibilidades a la hora de elegir distribuciones, *window manager* y configuraciones pero, por fortuna, se ha tratado de homogeneizar los pasos para la configuración del sistema de tal forma que, por su sencillez, probar con ésta u otra configuración puede convertirse incluso en un hobby o, ¿quién sabe?, quizá una forma de arte. ■



TRATAMIENTO DIGITAL DE IMÁGENES (IV)

Enrique Coiras

La potencia de los ordenadores domésticos permite realizar funciones de tratamiento de imágenes que hasta hace poco estaban reservadas exclusivamente a las estaciones de trabajo y ordenadores dedicados.

En esta cuarta entrega se estudiarán las transformaciones geométricas aplicadas al tratamiento de imágenes y algunos métodos de interpolación

TRANSFORMACIONES GEOMÉTRICAS

Unas de las operaciones que más comúnmente se aplican al manipular imágenes digitales son las de traslación, rotación y escalado. Estas, así como todas aquellas operaciones que signifiquen una modificación de la disposición de la información de la imagen con respecto a su sistema de coordenadas original, se engloban bajo la denominación de transformaciones geométricas. Las distorsiones de las imágenes, como el conocido *morphing* por ejemplo, se incluyen también bajo esta denominación.

En general, una transformación geométrica pondrá en correspondencia un punto (x, y) de la imagen final con un punto (x0, y0) de la imagen original. Ello hace posible que la implementación de las rutinas de transformación geométrica se pueda enfocar de dos maneras distintas : o bien hacemos nuestra rutina para que vaya recorriendo todos los puntos de la imagen inicial y devuelva los puntos de la imagen final correspondientes, o bien para que recorra la imagen final y devuelva los puntos de la imagen inicial que les corresponden. El primer enfoque puede parecer el más lógico y, sin embargo, es el menos práctico de los dos. Al recorrer la imagen inicial es posible que haya puntos de la imagen final a los que no les llegue ninguno de los puntos que hayamos transformado. Y también puede ocurrir que puntos que nos hayamos molestado en calcular caigan fuera de la imagen final, con lo que habremos perdido el tiempo inútilmente. Por ello es preferible ir tomando cada uno de los puntos de la imagen final y aplicarle la transformación inversa a la que en realidad queremos realizar.

La transformación más sencilla es la traslación, que consiste básicamente en desplazar la imagen según un vector, es decir, como si la moviéramos sobre una mesa sin hacerla girar. La formulación matemática de esta operación es la siguiente :

$$x = x0 + Dx;$$
$$y = y0 + Dy;$$

Pero normalmente se suele utilizar la notación matricial para estas operaciones. Para la transformación anterior la operación se notaria así :

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & Dx \\ 0 & 1 & Dy \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x0 \\ y0 \\ 1 \end{pmatrix}$$

La coordenada "extra" (el 1) que se añade a los vectores se utiliza para poder incluir la traslación dentro de la notación matricial de las aplicaciones lineales.

Lo que nos interesa es realizar la transformación inversa, así que la operación a realizar será la siguiente :

$$\begin{pmatrix} x0 \\ y0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -Dx \\ 0 & 1 & -Dy \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

que devolverá las coordenadas del pixel de la imagen inicial en función de la de las coordenadas del pixel destino.

Con esta y con todas las demás opera-

```
#define OUT_PIXEL 0

int puntovalido (imagen img, punto p)
{
  return ((p.x >= 0) && (p.x < img.sizex)
        && (p.y >= 0) && (p.y < img.sizey));
}

unsigned char imgval (imagen img, punto p)
{
  if (puntovalido (img, p))
    return (img.dat [p.y][p.x]);
  else
    return (OUT_PIXEL);
}
```

Figura 1.

ciones, existirán ocasiones en las que las coordenadas del punto inicial se encuentren fuera de la imagen. Por ello suele ser útil crearse una función que, según las coordenadas de un punto, devuelva el valor del pixel correspondiente en la imagen, si es que este se encuentra dentro de la misma, o un valor constante determinado si el punto cae fuera de rango. Eso es precisamente lo que hace la función *imgval* de la figura 1.

Análogamente podemos definirnos una función que nos modifique el valor de pixel del punto de una imagen sólo si este se encuentra dentro de sus fronteras : (figura 2).

Antes de seguir, es necesario hacer otra precisión. Aunque cuando nos referimos a un punto de una imagen lo hacemos mediante sus coordenadas, en realidad no es esa su posición real, sino la de su esquina superior izquierda (según el convenio más extendido). El centro del punto (x, y) se encuentra en realidad en la posición (x + 1/2, y + 1/2), y es allí donde debemos mirar cuando queramos obtener su valor. Este refinamiento puede parecer superfluo, pero conviene tenerlo en cuenta aunque sólo sea por mantener la coherencia

```
void setimgval (imagen img, punto p, unsigned
char pixelval)
{
  if (puntovalido (img, p))
    img.dat [p.y][p.x] = pixelval;
}
```

Figura 2.

```
typedef struct {
  x, y : float;
} vector;

void traslacion (imagen img, vector v,
imagen imgout)
{
  int y, x, y0, x0;

  for (y = 0; y < imgout.sizey; y ++)
    for (x = 0; x < imgout.sizex; x ++) {
          x0 = (int) ((float) x - v.x + 0.5);
          y0 = (int) ((float) y - v.y + 0.5);

  imgout.dat [y][x] = imgval (img, defpunto (y0, x0));
    }
}
```

Figura 3.

matemática de nuestro desarrollo.

En la figura 3 podemos ver como sería la rutina de traslación según un vector v = (Dx, Dy) :

Otra transformación geométrica básica es la de escalado. En este caso lo que se hace es multiplicar las dimensiones de la imagen original por un cierto factor. Si el factor de escala es mayor que uno la imagen final será una ampliación de la original, y si es menor será una reducción. En realidad los factores de escala son dos, uno para las coordenadas x y otro para las y, pero en ampliaciones y reducciones normales los dos son iguales. Si tomamos el factor de escala en las x mayor que el de las y, la imagen final aparecerá "estirada" en sentido horizontal, mientras que si lo hacemos al revés lo será en sentido vertical. Las fórmulas de transformación para el escalado se muestran en la figura mientras que con notación matricial queda :

$$\begin{pmatrix} x0 \\ y0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1/Sx & 0 & 0 \\ 0 & 1/Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

El centro de esta transformación es el origen de la imagen inicial, es decir, el

$x = Sx * x0 \quad <-> \quad x0 = 1/Sx * x$

$y = Sy * y0 \quad <-> \quad y0 = 1/Sy * y$

Figura 4.

$x - Cx = Sx * (x0 - Cx0) \quad <-> \quad x0 = (x - Cx) * 1/Sx + Cx0$

$y - Cy = Sy * (y0 - Cy0) \quad <-> \quad y0 = (y - Cy) * 1/Sy + Cy0$

Figura 5.

punto (x0 = 0, y0 = 0), que se corresponde con el punto final (x = 0, y = 0). Esto significa que lo que estamos haciendo al aplicarla es ampliar o reducir la zona alrededor de la cual esta el origen de coordenadas, cuando lo que normalmente nos interesará es ampliar una zona determinada de la imagen, centrada en un punto genérico (Cx0, Cy0). Y no sólo eso, sino que además es deseable que el punto sobre el que hemos centrado la transformación aparezca en el centro de la imagen final, esto es, el punto (Cx = imgout.sizex/2, Cy = imgout.sizey/2). Para conseguir hacer todo esto debemos utilizar la función de traslación antes vista. Tendremos que hacer dos traslaciones, una para centrar el punto de interés en la imagen inicial y otra para llevar el origen de coordenadas al centro de la imagen final. El resultado aparece en el listado 5, donde (Cx, Cy) es el punto de la imagen final que tomaremos como centro de coordenadas y (Cx0, Cy0) es el punto de la imagen original en el que queremos centrar el escalado.

Para obtener la notación matricial de la expresión anterior debemos calcular el producto de las dos matrices de traslación y la de escalado para obtener la matriz resultado :

Ahora ya podemos implementar la rutina de escalado incluyendo las dos traslaciones (figura 6)

Una función bastante extendida es la

```
void escalado (imagen img, vector C0, vector S,
vector C,
          imagen imgout)
{
  int y, x, y0, x0;

  if ((S.x == 0.0) || (S.y == 0.0)) {
    printf ("ESCALADO : Factores de escala no válidos!\n");
    exit (-1);
  }
  for (y = 0; y < imgout.sizey; y ++) {
    y0 = (int) (((float) y - C.y) / S.y + C0.y + 0.5);

    for (x = 0; x < imgout.sizex; x ++) {
      x0 = (int) (((float) x - C.x) / S.x + C0.x + 0.5);

      imgout.dat [y][x] = imgval (img, defpunto (y0, x0));
    }
  }
}
```

Figura 6.

```
void zoom (imagen img, punto a0, punto b0, imagen imgout)
{
  vector S, C, C0;

  if (((b0.x - a0.x) == 0) || ((b0.y - a0.y) == 0)) {
    printf ("ZOOM : Ventana no válida!\n");
    exit (-1);
  }

  /* Calculamos vector de escalado y centros de traslación */

  S.x = (float) abs ((double) imgout.sizex / (double) (b0.x - a0.x));
  S.y = (float) abs ((double) imgout.sizey / (double) (b0.y - a0.y));

  if (S.x > S.y)
    S.x = S.y;
  else
    S.y = S.x;

  C0.x = (float) (a0.x + b0.x) / 2.0;
  C0.y = (float) (a0.y + b0.y) / 2.0;

  C.x = (float) imgout.sizex / 2.0;
  C.y = (float) imgout.sizey / 2.0;

  /* Llamamos a la rutina de escalado */

  escalado (img, C0, S, C, imgout);
}
```

Figura 7.

de hacer el *zoom* de una pequeña ventana de la imagen original, definida por las coordenadas de sus vértices superior izquierdo (x0a, y0a) e inferior derecho (x0b, y0b). A partir de esos puntos podemos obtener los valores de Sx, Sy, Cx0 y Cy0. El valor de Cx y Cy serán las coordenadas del centro de la imagen final (imgout.sizex/2, imgout.sizey/2), igual que antes. Para asegurar que la imagen escalada no sufre distorsiones por diferencias de aspecto entre los tamaños de la ventana y de la imagen final asignamos el mismo valor a Sx y

$$\begin{pmatrix} x0 \\ y0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & Cx0 \\ 0 & 1 & Cy0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1/Sx & 0 & 0 \\ 0 & 1/Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & -Cx \\ 0 & 1 & -Cy \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1/Sx & 0 & Cx0 - Cx/Sx \\ 0 & 1/Sy & Cy0 - Cy/Sy \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$x - Cx = (x0 - Cx0) * \cos(A) - (y0 - Cy0) * \sin(A)$$

$$y - Cy = (x0 - Cx0) * \sin(A) + (y0 - Cy0) * \cos(A)$$

Figura 8.

```
void rotacion (img, vector C0, float A, vector C, imagen imgout)
{
  int y, x, y0, x0;

  float cA, sA, dx, dy;


  cA = (float) cos (A);

  sA = (float) sin (A);


  for (y = 0; y < imgout.sizey; y ++)

    for (x = 0; x < imgout.sizex; x ++) {

      dx = (float) x - C.x;

      dy = (float) y - C.y;


      x0 = (int) ( dx * cA + dy * sA + C0.x + 0.5);

      y0 = (int) (-dx * sA + dy * cA + C0.y + 0.5);


      imgout.dat [y][x] = imgval (img, defpunto (y0, x0));

    }
}
```

Figura 9.

Sy (tomando el que implique menor aumento), la función se halla en el listado 7.

La transformación de rotación se maneja de forma análoga a las anteriores. En este caso las fórmulas son :
donde A es el ángulo que queremos hacer girar la imagen, tomado en el sentido de las agujas del reloj (ya que el eje y va hacia abajo en la imagen). La inversa de la transformación de rotación de ángulo A es la rotación de ángulo -A. Teniendo esto en cuenta, así como el hecho de que cos (-A) = cos (A) y sin (-A) = -sin (A) podemos escribir la transformación completa en notación matricial:

En esta transformación las traslaciones se utilizan simplemente para restaurar la posición del centro de giro, y lo corriente es que Cx = -Cx0 y Cy = -Cy0. La implementación de esta transformación puede verse en listado 9.

$$\begin{pmatrix} x0 \\ y0 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(A) & \sin(A) & -Cx*\cos(A) - Cy*\sin(A) + Cx0 \\ -\sin(A) & \cos(A) & Cx*\sin(A) - Cy*\cos(A) + Cy0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

```
typedef struct {
  x, y : float;
} fpunto;

unsigned char imgval_bilin (imagen img, fpunto p)
{
  int xe, ye;
  float xfr, yfr;
  float pixval;

  /* Coordenadas enteras */

  xe = (int) p.x;
  ye = (int) p.y;

  /* Partes fraccionarias */

  xfr = p.x - (float) xe;
  yfr = p.y - (float) ye;

  /* Aportaciones de los cuatro vecinos */

  pixval = (float) imgval (img, defpunto (xe, ye)) * (1 - xfr) * (1 - yfr);
  pixval += (float) imgval (img, defpunto (xe + 1, ye)) * xfr * (1 - yfr);
  pixval += (float) imgval (img, defpunto (xe, ye + 1)) * yfr * (1 - xfr);
  pixval += (float) imgval (img, defpunto (xe + 1, ye + 1) * xfr * yfr;

  return ((unsigned char) pixval);
}
```

Figura 10.

Figura 1 : Una imagen y el resultado de trasladarla según un vector v = (Dx, Dy). Obsérvese como algunos puntos de la imagen resultado han caído fuera de la imagen original.

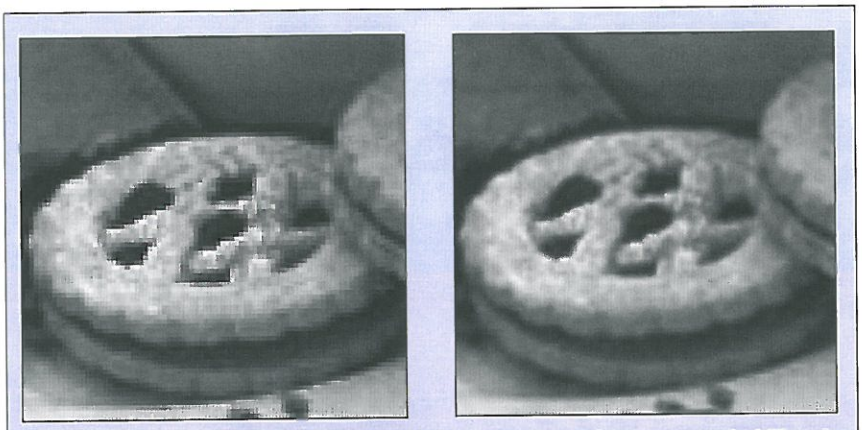


Figura 3. Comparación de la interpolación de vecino más cercano y la bilineal. En la primera puede apreciarse claramente el efecto mosaico.

MÉTODOS DE INTERPOLACIÓN

Generalmente, al calcular la transformación inversa de un punto de la imagen final obtenemos unas coordenadas que no caen exactamente en el centro de un punto de la imagen inicial. Debemos, por tanto, estimar a partir de los *pixels* más cercanos a esas coordenadas cual será el nivel de gris que asignaremos al punto de la imagen destino.

Hasta ahora hemos utilizado el criterio de escoger al *pixel* más cercano al punto calculado. Este tipo de aproximación, llamada *interpolación del vecino más cercano*, puede darnos errores de hasta *sqrt (2) / 2* unidades, además de crear el conocido efecto de mosaico, cuando se amplía mucho la imagen inicial. Para evitar esto existen otros métodos de interpolación que tienen en cuenta los cuatro o los nueve *pixels* más cercanos al punto en cuestión según determinadas funciones peso para cada uno de ellos.

El método de *interpolación bilineal* utiliza los valores de los cuatro pixels entre los que cae el punto para estimar su nivel de gris asignando pesos a las distancias que los separan del mismo. Podemos crearnos una nueva función *imgval* que utilice este método de interpolación para devolver valores de *pixels* (listado 10).

Y ahora podemos incluir la interpolación bilineal en las rutinas de transformación geométrica explicadas antes sin más que sustituir la función *imgval* antigua por esta otra. En la figura 3 puede verse fácilmente la mejora que supone este método sobre el primero.

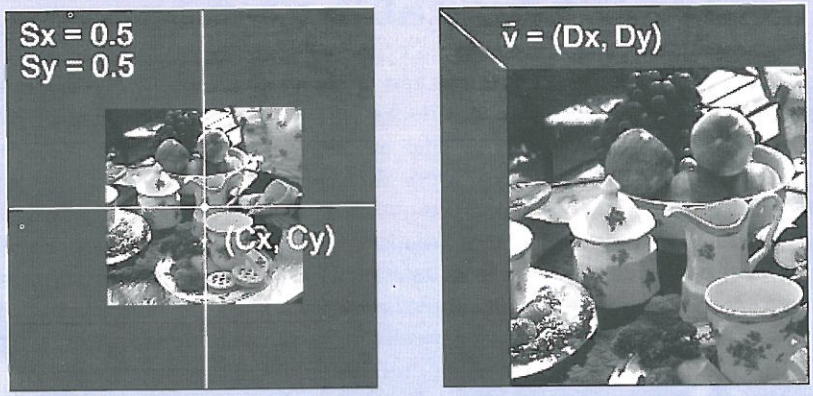


Figura 2. Resultados de rotar y escalar la imagen de la Figura 1. El origen de ambas transformaciones se ha tomado en el centro de la imagen.

Update now!

WATCOM

Compiladores para lenguajes profesionales
- Lenguaje C/C++V.10.0 CD ROM
- Lenguaje Fortran 77/32 Bits v 9.5

Ambiente de desarrollo visual (Front End)
- Lenguaje OS/2 Rexx
- VX-REXX v.2.1 Standard Edition
- VX-REXX v.2.1 Client/Server

Bases de datos relacionales (llamada SQL) v 4.0
- Plataformas: Windows, NT, OS/2, DOS y Netware
- Usuarios: Standalone - Network Server
- Lenguaje: C/C++

RAIMA

Raima Database Manager: Base de Datos
- Lenguaje: C
- Sistema Operativo: DOS, Windows, OS/2, UNIX System V, Berkeley 4.2 AIX, Sun OS, SCO, QNX, YULTRIX y VMS
- Plataformas: DOS, OS/2
- Modalidades: Module y System
- Royalty Free

VELOCIS

Velocis: Base de datos. Tecnología Cliente/Servidor
- Lenguaje C/C++
- Sistema Operativo: Windows, NT y UNIX
- Plataformas: Windows, NT, OS/2, Netware 386 NLM, OS/2 y UNIX
- Velocidad de acceso 1/3 más rápido que otras B/Datos

PHARLAP

Gestiona la memoria optimizando los límites de todas las versiones DOS

SEQUITER Software Inc.

Librerías de programas para trasladar aplicaciones de Dbase a C/C++
- CodeServer v.5.1 Cliente/Servidor

OFERTA ESPECIAL

Para los lectores de SÓLO PROGRAMADORES

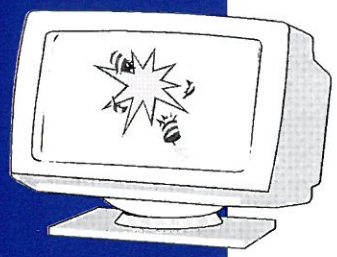
Infórmate llamando:
Tel.: 485 13 04
Fax: 485 14 29

CMR Competitive Manufacturing Research

C. Lope de Vega, 21 bjs.
08005 Barcelona

EL TÚNEL DE ESTRELLAS

Pedro Antón



Cualquier demo usa gran cantidad de bytes para almacenar en tablas todo lo que pueda ser precalculado, es decir, todas aquellas secuencias de bytes que se repitan de forma cíclica, y no varíen de valor, deben ser guardadas en tablas. El porqué de esto es evidente, la velocidad. Pero esto se verá mejor con el más típico ejemplo, el ejemplo de la tabla de senos y cosenos: Los senos y cosenos son una serie de datos que siempre se repiten de igual forma, quiere decir esto, que el seno de cero grados, es siempre cero, y es coseno de cero grados es siempre uno. El 99% de las demos hacen uso de una tabla de senos y cosenos, ya que es seguro que alguno de los efectos que se muestre en la demo necesite de estos tan preciados valores.

Ahora bien, a la hora de crear una tabla se han de tener en cuenta un par de cosas: la precisión de ésta y la cantidad de datos de la tabla. Se debe diferenciar bien entre precisión y cantidad de datos, al decir precisión se refiere a la exactitud del dato en cuestión, si se almacenan coordenadas de pantalla, la precisión es del 100%, al almacenar el punto (160,100) no se obtiene ningún problema de precisión ya que se almacenará exactamente ese punto y no uno similar, pero esto no ocurre así al almacenar, en el anterior ejemplo, los valores de los senos y cosenos. El seno de 52 grados es exactamente 0.788010753607, almacenar datos de este estilo no será un problema para la mayoría de los programadores, pero operar con este tipo de datos puede resultar nefasto para la velocidad del efecto deseado. Así pues siempre se almacenarán los datos en punto fijo, y aquí es donde se deberá tener en cuenta la precisión:

Dato indica el valor a almacenar en punto fijo, var indica un valor tipo con el que será operado el valor almacenado en punto fijo, cuanto más pequeños sean estos datos, menor precisión será necesaria. Al pasar un número a punto fijo se debe multiplicar dicho número por dos elevado al número de bits en punto fijo que deseemos. Después de realizar los cálculos pertinentes con el dato en cuestión, se realizará un desplazamiento lógico a la derecha tantas veces como la cantidad de bits en punto fijo elegida.

En estos datos se aprecia la precisión obtenida al coger mayor cantidad de bits de punto fijo para construir las

```
Dato : 0.788010753607
var  : 120
Dato*var=94.5612904328

Punto fijo 2 bits = dato*    4 =    3 * var sar  2 = 90
Punto fijo 4 bits = dato*   16 =   12 * var sar  4 = 90
Punto fijo 6 bits = dato*   64 =   50 * var sar  6 = 93
Punto fijo 8 bits = dato*  256 =  201 * var sar  8 = 94
Punto fijo 10 bits = dato* 1024 = 806 * var sar 10 = 94
Punto fijo 12 bits = dato* 4096 = 3227 * var sar 12 = 94
Punto fijo 14 bits = dato*16384 = 12910 * var sar 14 = 94
Punto fijo 16 bits = dato*65536 = 51643 * var sar 16 = 94
```

El efecto "tunel de estrellas" es muy útil pues en él se hace un gran uso de las tablas

En este artículo se explicará el efecto del túnel de estrellas, aunque se trata de un efecto muy sencillo, tiene una vital importancia en esta serie de artículos debido al tremendo uso que hace las tablas.

tablas y se observa con claridad como a partir de 8 bits en punto fijo la precisión no varía, con el dato elegido. Se ha elegido 120 como dato, porqué se trata de un valor tipo en las demos, normalmente los valores de cálculo estarán entre 0 y 320. La mayoría de los programadores de demos usan 8 bits de punto fijo.

Otro dato a tener en cuenta a la hora de elegir el número de bits para hacer tablas en punto fijo, es la posibilidad de obtener un desbordamiento en los cálculos, lo que puede ocurrir, por ejemplo si se trabaja con más de ocho bits de punto fijo y con enteros. Veamos un caso práctico en el que ocurrirá esto:

```
sin (87) = 0.998629534755
En punto fijo de 10 bits = 1024 * var = 122880
```

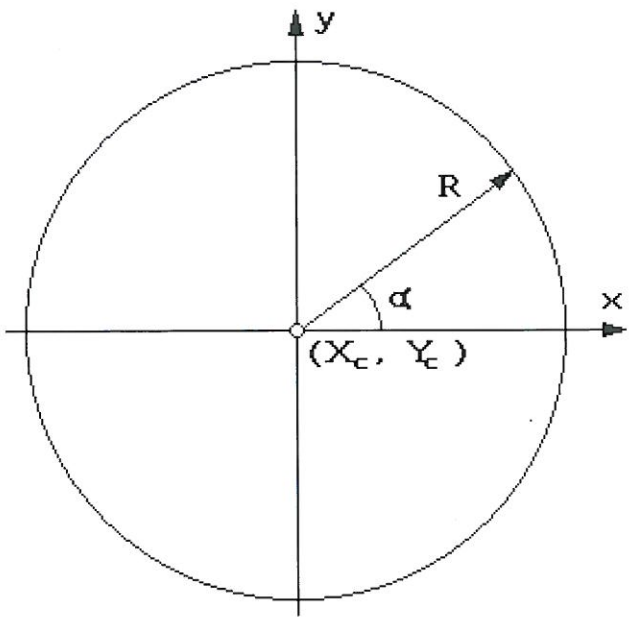
Si se está trabajando con enteros este valor no puede ser almacenado en un registro de 16 bits, por lo tanto, para usar tablas en punto fijo mayores de ocho se deben emplear los registros extendidos del 386, que ya son de 32 bits, en lenguajes de alto nivel, se usarán tipos de variables de 32 bits, pero esté seguro de que eso se paga muy caro en la velocidad del efecto.

Por último, explicar la importancia del número de elementos de una tabla. En realidad lo que se trata de explicar aquí es la ventaja de usar un byte como puntero a una tabla. Si a un byte con valor 255 se le incrementa en uno pasa a valer cero, y eso es genial para determinado tipo de tablas, por ejemplo una tabla de senos y cosenos, está claro que cero grados menos uno no es menos un grado sino 359 grados, y eso es exactamente lo que ocurre con un byte, con la salvedad de que una circunferencia no se define con 255 grados, no obstante, se definirá con 255 valores a cada grado real le corresponderán 1.40 grados de estos grados "demoeros". Sin embargo en el código adjunto la tabla donde se almacena el valor de la tabla de senos es:

SinTable : array [0..449] of integer;

Donde el valor 449 es el resultado de 359+90. El por qué de ese valor es pura trigonometría: el seno de un ángulo es igual al coseno del mismo ángulo más noventa grados. En este caso no se define una circunferencia en 255 pasos, porque el número de pasos

```
{ Hace un degradado entre los valores 16 y 32
dados unos valores     }
{ iniciales de RGB. El color inicial, es el más oscu-
ro del degradado. }
Procedure MakeDegradated (InitR,InitG,InitB:Byte);
Var
  CntColor : Byte;
{ No deja que de 0 pase a 255.}
  Procedure MyDec (Var Val:Byte;Inc:Byte);
  Begin
    If Val>Inc then Dec (Val,Inc)
            else Val:=0;
  End;
  Procedure SetColor(Col,R,G,B:Byte); assembler;
  Asm
    cli
    mov dx,03C8H
    mov al,Col
    out dx,al
    inc dx
    mov al,r
    out dx,al
    mov al,g
    out dx,al
    mov al,b
    out dx,al
    sti
  End;
Begin
  For CntColor:=32 downto 16 do
    Begin
      SetColor (CntColor,InitR,InitG,InitB);
      MyDec (InitR,4);
      MyDec (InitG,4);
      MyDec (InitB,4);
    End;
End;
```



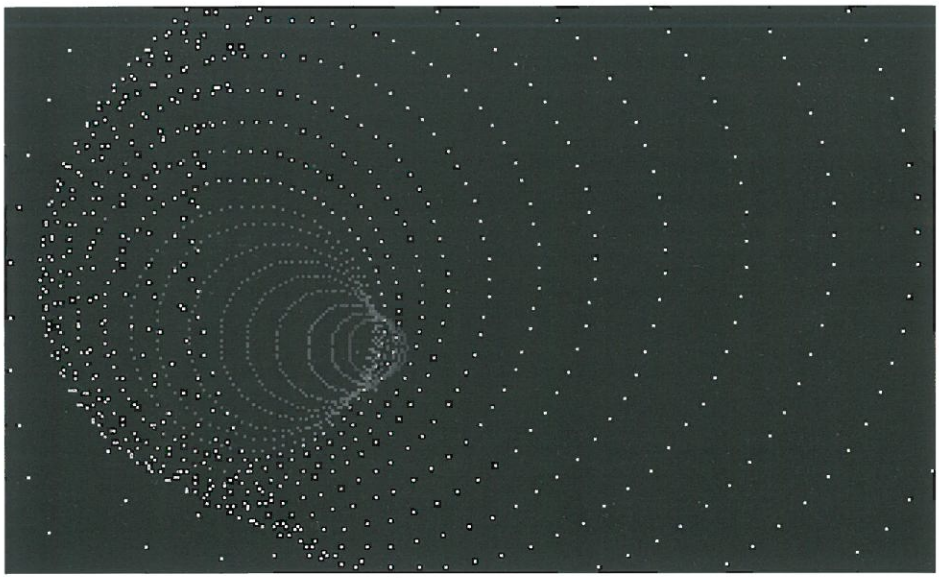
Cartesianas:

$$(x - X_c)^2 + (y - Y_c)^2 = R^2$$

Paramétricas:

$$x = X_c + R \cos(\alpha)$$
$$y = Y_c + R \operatorname{sen}(\alpha)$$
$$\alpha \in (0,360)$$

Esta ecuación, con pequeñas diferencias, será utilizada muy a menudo.



El efecto del túnel de estrellas resulta muy vistoso a la vez que sencillo.

usados para definir una circunferencia será el número de puntos que aparezcan en pantalla definiendo cada circunferencia.

Este artículo parece una clase de matemáticas ¿verdad?, bien, hace unos años leí unos documentos de rotaciones en 3D, eliminar planos ocultos y otra serie de cosas, estos documentos los escribió Lithium de VLA (aquellos de vosotros que tengáis acceso a Internet, habréis visto más de un artículo de VLA). Al principio de esos documentos ponía: "REAL PROGRAMMERS ARENíT AFRAID OF MATH" que viene a ser algo así como "los programadores de verdad no tiene miedo de las matemáticas" así es que, continuemos con el artículo y un poquito más de matemáticas.

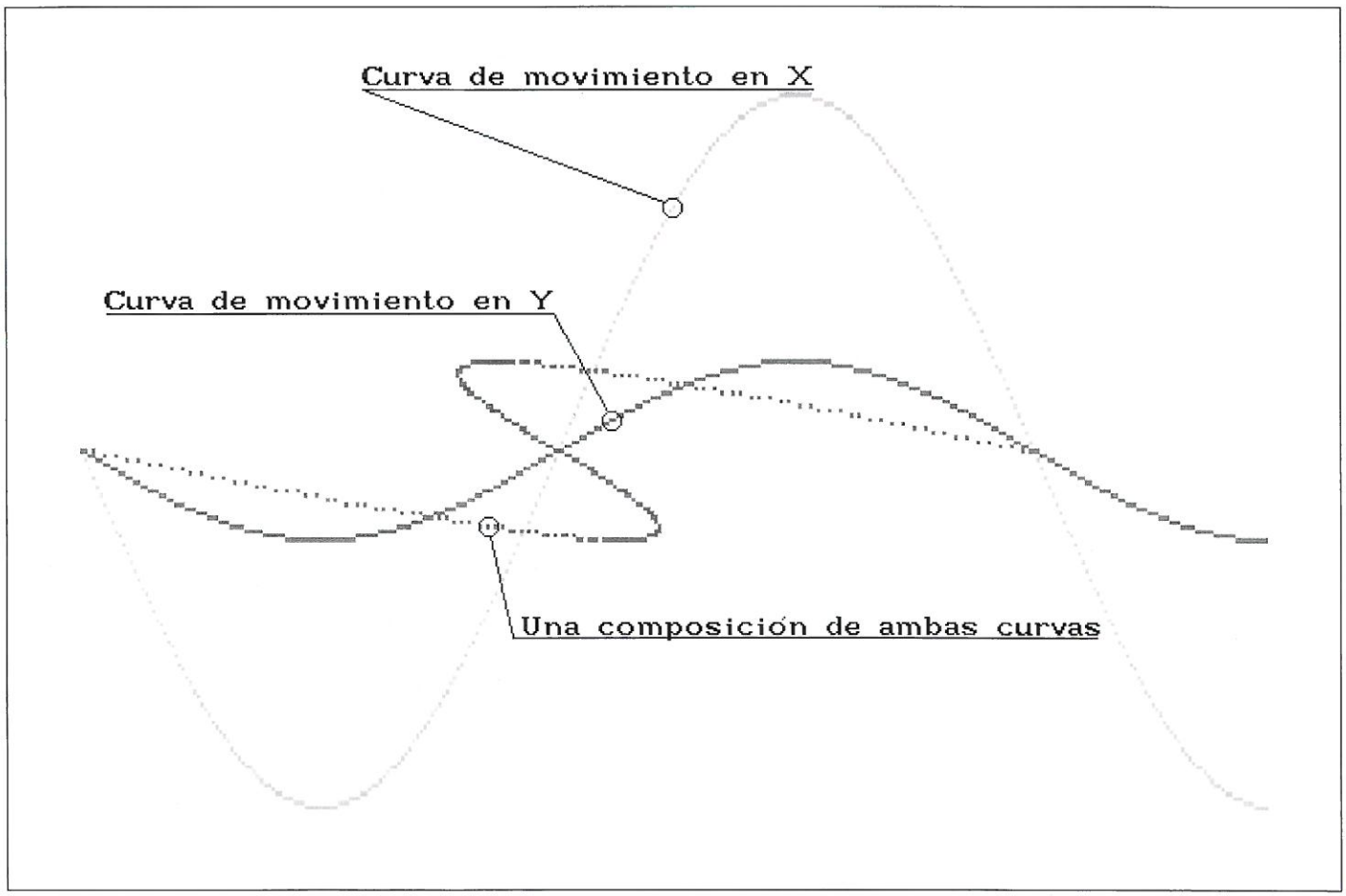
LA ECUACIÓN DE LA CIRCUNFERENCIA

Esta ecuación con distintas variaciones es muy usada. Siguiendo con las matemáticas, para conseguir este efecto es primordial conocer las ecuaciones de una circunferencia.

En este efecto se usarán las coordenadas paramétricas para dibujar los puntos de cada circunferencia, en el código adjunto únicamente se dibujan puntos, tal como hizo Future Crew en Unreal, no obstante, es posible dibujar líneas, o rellenar las circunferencias.

EL MOVIMIENTO

El agujero no puede estar parado. Este efecto se puede crear de múltiples



Como puede observarse, las curvas X e Y se repiten.

formas, pero ésta es sin duda la más rápida, y quizás la más espectacular. Lo más importante que se debe tener en cuenta en este efecto es que el agujero ha de moverse constantemente. En el código adjunto, los valores de movimiento en los ejes X e Y son las tablas:

```
XSinMov : array [0..255] of integer;
YSinMov : array [0..255] of integer;
```

El movimiento consiste en dos curvas que se unen por sus extremos y se recorren de forma ciclica

Se debe observar en estás tablas que la cantidad de datos que almacenan es exactamente un byte, en contraposición con la tabla mencionada anteriormente.

LA PALETA

El agujero usa 16 colores para dar sensación de profundidad. En el artículo anterior el fuego usaba una gran cantidad de colores y el código usaba un fichero FIREPAL.INC donde iban definidos los colores de la paleta, en este artículo sólo usamos 16 colores, así es que se construirá la paleta con unas líneas de código:

La utilización de tablas facilita y acelera los cálculos a realizar para cada frame

El degradado queda entre los colores 16 y 32, esos colores podrían haber estado entre otros límites diferentes. Os preguntaréis ¿y por qué están exactamente ahí y no entre 0 y 15?. Responderé a esa pregunta, normalmente en mi código me gusta dejar los primeros 16 colores para posibles máscaras o colores base, y comienzo a realizar los degradados a partir del color 16. Cada programador puede elegir donde pone los colores a su antojo, no existe ningún tipo de norma para este tipo de cosas, se comenta esto únicamente para disipar posibles dudas.

EL BUFFER

Se usará memoria dinámica. Como en casi todos los efectos usaremos un buffer, en este caso, de 64000 bytes (320x200). No se reserva la memoria definiendo un *array* ya que se puede sobrepasar, los 65535 bytes del segmento de datos, en cuyo caso se deberá definir el array en una unidad, se reserva la memoria usando memoria dinámica, memoria que podemos liberar en el momento que deseemos. Al solicitar al sistema operativo 64000 bytes de memoria dinámica se obtiene un segmento y un *offset* cero, pero es importante saber que sólo ocurre esto al reservar una cantidad de memoria muy cercana al segmento, de lo contrario se obtendrá un *offset* distinto de cero.

EL BUCLE PRINCIPAL

Como ya habréis notado, este efecto destaca por el uso de las tablas. Estas tablas son las que hacen posible que este efecto sea rápido, fíjese en la cantidad de cálculos por *frame* que se debería hacer si no se usasen tablas, de aquí se desprenden un par de normas que todos debemos tener muy en cuenta a la hora de preparar una demo:

* Precalcula tantos valores como sea posible.
* No calcules el mismo valor dos veces. Almacénalo.

Los pasos del programa serán:

* Reservar memoria dinámica para el buffer, aunque más adelante se verá que en este efecto, se podía haber conseguido bastante más velocidad usando el modo *Xmode*. Almacenar en una variable el segmento donde se encuentra el buffer reservado, (recuerde la segunda norma).
* Precalcular las tablas. Tanto la de movimiento como la de senos.
* Crear el degradado que se va a usar para conseguir mayor sensación de profundidad, escribiendo las circunferencias de menor tama–o de color más oscuro.
* Asegurarnos de limpiar (rellenar con ceros) la zona de memoria donde está el buffer. Este paso se debe hacer siempre cada vez que ponemos un *frame* en la pantalla.
* Y ahora viene el "truquillo" de este efecto, el movimiento, ya se ha visto la ecuación paramétrica de una circunferencia, donde se puede ver la facilidad para dibujar una circunferencia en el ordenador usando esta ecuación, pero ¿y el movimiento?. El movimiento consiste en realizar dos curvas, una sobre el eje x y otra sobre el eje y, que se unan en sus extremos, que serán recorridas de forma cíclica, empezando a poner circunferencias siempre desde la más lejana, y teniendo en cuenta que la distancia que separa dos circunferencias es menor si están más lejos.

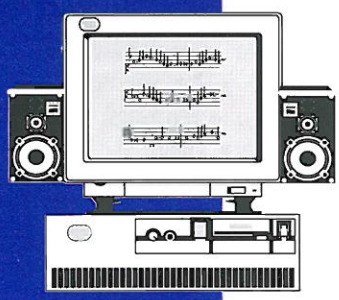
Como se aprecia en el gráfico, tanto la curva de movimiento en X como la de movimiento en Y se repiten, son de período 255, por lo tanto haciendo una composición adecuada (no la del gráfico) de ambas se obtiene un movimiento senoidal en todas las direcciones bastante agradable.

Una vez dibujadas las circunferencias, teniendo en cuenta el color, la distancia entre ellas y el centro de cada una de ellas, ya estará listo el *frame*, y se procederá a volcarlo en la pantalla en el momento exacto en que finalice el retrazo vertical.

Se limpia el buffer para preparar el siguiente *frame*, y se repite la operación incrementando el origen de la última circunferencia, según la trayectoria deseada o precalculada. Con esto habremos conseguido el efecto deseado. ■

TEMPORIZACIÓN Y EFECTOS EN LOS MOD

Luis Crespo



En la entrega anterior se explicó cómo mezclar varias voces a distintas frecuencias para luego enviarlas a un DAC único. Pero es necesario saber en cada momento qué nota debe sonar en cada canal.

En este artículo se explicarán muchas cosas que tratarán sobre la interpretación de la partitura de un módulo. Por ello es recomendable tener los conceptos frescos, y no estaría de más releer el artículo sobre el formato de los ficheros MOD que se encuentra en el número 6 de la revista, para tener claro qué es un patrón, un canal, la secuencia, el periodo, etc.

Recordemos como funcionaba básicamente el reproductor de MODs del artículo anterior: la rutina *SongPlay* del fichero SONGS.PAS prepara un doble buffer para que suene continuamente por el DAC del dispositivo de sonido. La rutina que es llamada al final de cada buffer se encarga de llenarlo con el resultado de la mezcla de los distintos canales, llamando a la rutina *NextTick*. Esta rutina es la que consulta qué digitalización está sonando por cada canal y a qué frecuencia, y consecuentemente genera un buffer con la mezcla.

TEMPORIZACIÓN

Al mismo tiempo hay que interpretar la partitura e ir determinando en cada momento qué instrumento y qué nota debe sonar en cada canal. O lo que es lo mismo: qué digitalización y a qué frecuencia, para que la rutina de mezcla genere el buffer correcto. Como ya se explicó, la base de tiempos por defecto de los MODs es 1/50 de segundo. Es decir, con un tempo de 6, por ejemplo, entre una nota y la siguiente hay 6/50 de segundo, y en general un tempo T significa que entre nota y nota transcurren T unidades de tiempo o *ticks*.

Como el tamaño de un buffer se calcula de forma que entre un buffer y el siguiente transcurre exactamente un *tick*, para determinar en qué momento debe pasarse a la siguiente nota basta con utilizar un contador de *ticks*. Al principio de cada nota el contador es igual al tempo, y se decrementa tras cada *tick*. Cuando el contador vale 0, es el momento de cambiar de nota y hacer de nuevo que el contador sea igual al tempo. Como se lleva control de qué línea y patrón se está interpretando, no es difícil determinar cuál es la siguiente nota: simplemente se incrementa la línea y se obtiene la nueva nota de la partitura. Si la línea es mayor que 64 hay que pasar al siguiente patrón, y también hay que tener en cuenta si hay algún efecto de salto, como se verá más adelante.

Para cada canal del MOD hay que decodificar la nota. Es decir, en función de los datos de cada nota (divisor, volumen, instrumento, etc), hay que actualizar los datos del correspondiente DAC virtual, o lo que es lo mismo, los campos de la estructura TChannelInfo de cada canal. De esto se encarga la rutina *PlayNote*, teniendo en cuenta los siguientes factores:

- Si no se especifica el volumen, se utiliza el de la nota anterior.
- Si se especifica el periodo, además de cambiar la frecuencia de la nota, la digitalización correspondiente suena desde el principio.
- Si se especifica el instrumento hay que cambiar la digitalización.

Además, para cada nota hay que procesar el comando correspondiente, si es que hay alguno:

EFECTOS

Cuando se explicó el formato de los ficheros MOD ya se vieron algunos comandos, pero la lista de distintos comandos o efectos que puede haber

en una nota es bastante grande: como el formato MOD fue evolucionando con el tiempo, se han ido añadiendo nuevos comandos, algunos de los cuales son muy extraños y no se utilizan nunca. Por eso se presentarán los efectos por orden de importancia: primero se explicarán los utilizados más frecuentemente. En los recuadros 1 y 2 se puede ver una lista de comandos ordenados numéricamente. Como todos los programas de composición de MODs están en inglés, el comando se enuncia en su idioma original y a continuación se explica su significado.

Recordemos que los comandos van acompañados de un byte que es el parámetro, cuyo significado varia según cada comando. En algunos casos, el byte del parámetro se desglosará en los 4 bits altos y los 4 bajos, dando lugar a dos parámetros.

15: Set tempo/BPM. Si el parámetro es menor que 32, éste indica el nuevo tempo, si es mayor o igual, indica los nuevos BPM (ver el artículo sobre el formato MOD para una explicación de los BPM).

12: Set volume. Modifica el volumen por defecto de la nota. En las rutinas de ejemplo se decodifica este comando al cargar el fichero, colocándose en el campo "Volume" de la nota correspondiente.

13: Pattern break. La siguiente nota se leerá del siguiente patrón. El parámetro indica a partir de qué línea del patrón hay que empezar a leer, de una forma un tanto extraña: NotePos := 10*(Param SHR 4) + Param AND $F.

11: Position Jump. Este comando indica un salto en la secuencia de patrones. El parámetro es el número de secuencia de la que hay que obtener el patrón de la siguiente nota, empezando por la primera nota del patrón.

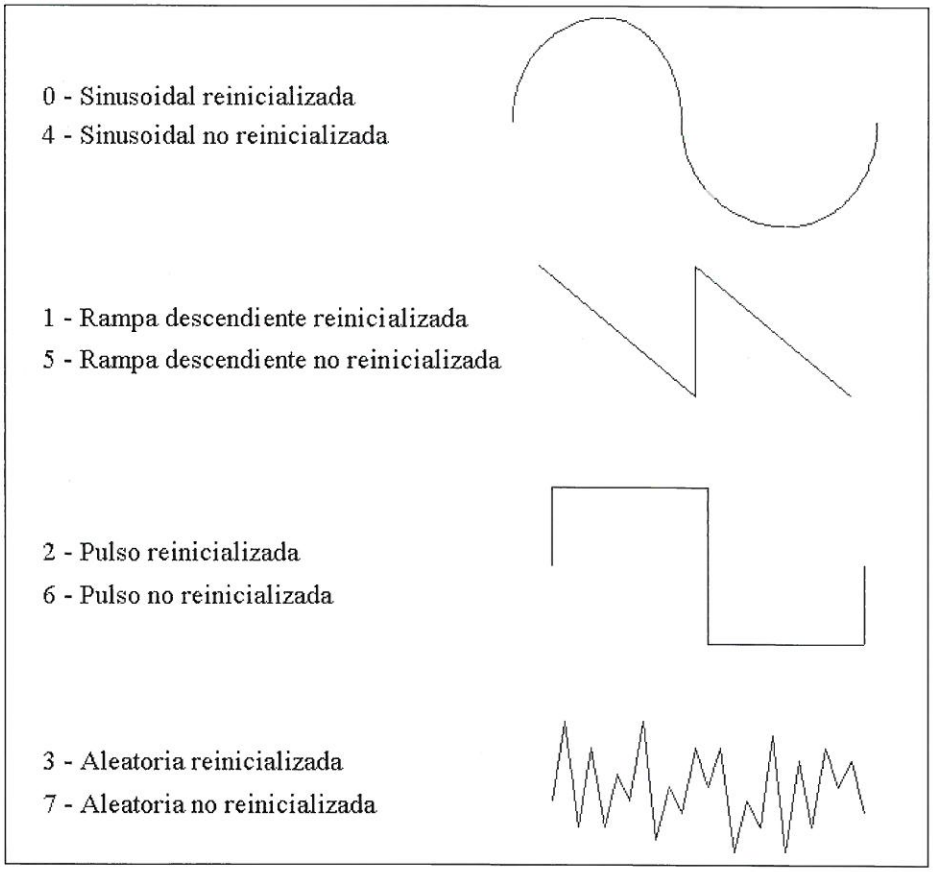
10: Volume slide. El volumen se decrementará o incrementará hasta la siguiente nota. El funcionamiento es el siguiente: si el número determinado por los 4 bits altos del parámetro es distinto de 0, se trata de un incremento, y si los 4 bits bajos determinan un número distinto de 0, se trata de un decremento (no es legal que ambos valores sean distintos de 0, ya que no se sabría si incrementar o decrementar). Tras cada *tick* el volumen se incrementa o decrementa el valor indicado, sin pasar nunca por encima de 64 o por debajo de 0. Por ejemplo, supongamos que el volumen es 32 y el tempo 5, y que el parámetro del efecto es 00100000 en binario. Esto significa que hay que incrementar el volumen en 2 en cada tick, de forma que al final de la nota el volumen valdrá 32 + 2* 5 = 42.

Para calcular la duración de una nota se utiliza un contador de ticks

1: Slide up. El periodo de la nota se decrementará en cada *tick* el valor indicado en el parámetro, controlando que el valor del periodo nunca sea menor que 113. Esto provoca un desplazamiento de la nota hacia tonos más agudos. Si el parámetro es 0, se utiliza el decremento del anterior slide up, para prolongar el efecto durante varias notas.

2: Slide down. Equivalente al efecto anterior, sólo que el periodo se incrementa en lugar de decrementarse, y nunca deberá superar el valor 856. El resultado es un desplazamiento hacia tonos más graves.

3: Slide to note. El periodo se incrementará o decrementará en cada *tick* el valor indicado en el parámetro, para alcanzar el periodo especificado en la nota. Por ejemplo, supongamos que el periodo de


0 - Sinusoidal reinicializada
4 - Sinusoidal no reinicializada
1 - Rampa descendiente reinicializada
5 - Rampa descendiente no reinicializada
2 - Pulso reinicializada
6 - Pulso no reinicializada
3 - Aleatoria reinicializada
7 - Aleatoria no reinicializada

Formas de onda para el vibrato y tremoto.

la nota anterior era 254 (La 4), que el periodo de la nota actual es 214 (Do 5), que el tempo es 6 y que el parámetro es 12. En cada *tick* se irá decrementando el periodo en 12, de modo que en el cuarto *tick* el periodo debería valer 206, pero como no puede pasarse por debajo de 214, el periodo valdrá 214 a partir de entonces. Si el parámetro es 0 se utiliza el valor del anterior slide to note, para alargar el efecto durante varias notas.

4: Vibrato. El periodo de la nota oscilará. La amplitud de la oscilación viene determinada por los 4 bits bajos del parámetro, de forma que para una amplitud A la nota oscilará A/16 semitonos. La velocidad de la oscilación la determinan los 4 bits altos del parámetro, de modo que para una velocidad V se completan V/64 ciclos de oscilación en cada tick. Si A o V son 0, se toma el valor antiguo para prolongar el vibrato.

9: Set sample offset. La digitalización debe reproducirse a partir de la muestra indicada en el parámetro. La posición de la muestra será P*256 (donde P es el parámetro).

5: Continue slide to note + Volume slide. Este efecto es en realidad la combinación de otros dos. Normalmente un efecto anula el anterior, así que con éste se puede mantener el anterior slide to note (con los parámetros anteriores) y al mismo tiempo hacer un volume slide. El parámetro es equivalente al del efecto de volume slide: si los 4 bits altos determinan un número distinto de 0, se trata de un incremento de volumen, y si los 4 bajos determinan un número distinto de 0, se trata de un decremento.

6: Continue vibrato + Volume slide. Muy similar al efecto anterior, sólo que el efecto que se mantiene con los parámetros anteriores es el vibrato.

7: Tremolo. El volumen oscilará de una forma muy similar a la del efecto de vibrato: el parámetro de velocidad funciona igual, mientras que la amplitud indica que el volumen debe oscilar entre Vol-A y Vol+A.

0: Arpeggio. Si el parámetro es 0, la nota suena sin efecto alguno. Si por el contrario no es 0, se toma P1 como los 4 bits altos y P2 como los 4 bajos, y se hace lo siguiente: durante el primer tercio de la línea se reproduce la nota original, durante el siguiente tercio se reproduce nota+P1 semitonos y durante el ultimo se reproduce nota+P2 semitonos.

Algunos de los efectos son muy extraños y no se utilizan nunca

Los slides se utilizan para cambios suaves en el volumen y la frecuencia

EFECTOS EXTENDIDOS

Como el comando o efecto de una nota puede tomar valores entre 0 y 15, sólo quedan dos efectos libres: el 8 y el 14. El comando 8 no se utiliza explícitamente en un MOD, pero a veces se utiliza para sincronizar la música con el programa, por ejemplo en una demo. Así que sólo queda el comando 14, pero como parece ser que hacían falta más efectos, se utilizó el siguiente truco: el comando 14 contiene efectos extendidos. Los 4 bits altos del parámetro indican el efecto extendido y los 4 bits bajos el parámetro del efecto extendido. Se tiene entonces los siguientes comandos, de nuevo ordenados por importancia o frecuencia de uso:

```
 0 : Arpeggio
 1 : Slide up
 2 : Slide down
 3 : Slide to note
 4 : Vibrato
 5 : Continue slide to note + Volume slide
 6 : Continue vibrato + Volume slide
 7 : Tremolo
 8 : No utilizado
 9 : Set sample offset
10 : Volume slide
11 : Position Jump
12 : Set volume
13 : Pattern Break
14 : Extended command
15 : Set tempo/BPM
```

Recuadro 1: Comandos.

14 - 1: Fineslide up. Muy similar al efecto slide up, pero el periodo sólo se decrementa en el primer tick.

14 - 2: Fineslide down. Equivalente al efecto anterior, sólo que el periodo se incrementa en lugar de decrementarse.

14 - 10: Fine volume slide up. Se incrementa el volumen en el primer tick, con el valor especificado en el parámetro.

14 - 11: Fine volume slide down. Equivalente al efecto anterior, decrementando el volumen.

14 - 5: Set finetune value. El parámetro modifica el afinamiento preciso por defecto de la nota, y viene en el mismo formato que en los instrumentos de la cabecera del fichero MOD.

14 - 4: Set vibrato waveform. El vibrato oscila de forma sinusoidal por defecto. El parámetro de este efecto indica de qué forma oscilará (ver figura 1). Además, si el parámetro está entre 0 y 3, la oscilación comenzará desde el principio para cada nueva nota, mientras que si está entre 4 y 7 se continuará desde la posición anterior.

14 -7: Set tremolo waveform. Equivalente al efecto anterior, pero se modifica la forma de onda de la oscilación del tremolo.

14- 9: Retrigger sample. Este efecto se utiliza para simular redobles de un instrumento, colocando el recorrido de la digitalización en el punto de inicio. El parámetro indica cada cuántos ticks se vuelve al principio de la digitalización.

14 - 12: Cut sample. Pone el volumen de la nota a 0 tras el número de ticks indicados en el parámetro.

14 - 13: Delay sample. Retrasa el inicio de la digitalización al *tick* indicado en el parámetro.

14 - 14: Delay pattern. Se retrasa la reproducción de todos los canales el número de líneas indicadas en el parámetro.

14 - 6: Loop pattern. Se repetirá una parte del patrón, de la siguiente forma: si el parámetro vale 0, el efecto indica la línea inicial de la repetición. Si no vale 0, indica el número de repeticiones, es decir, el número de veces que hay que volver al punto inicial. Si no se especifica el punto inicial, se toma la primera línea del patrón.

14 - 3: Glissando on/off. Si el parámetro es 0 se inhabilita el glissando, si es 1 se habilita. Por defecto el glissando está deshabilitado, y si se habilita, al ejecutar el comando 3 (slide to note), su parámetro indicará semitonos en lugar de incrementos de periodo.

14 - 0: Filter on/off. Utilizado en el Amiga para habilitar o deshabilitar el filtro de audio.

14 - 15: Invert samples. Tras un cierto tiempo especificado en el parámetro, las muestras de dentro del bucle se invierten, de forma que los valores positivos pasan a negativos y viceversa.

0 : Filter on/off
1 : Fineslide up
2 : Fineslide down
3 : Glissando on/off
4 : Set vibrato waveform
5 : Set finetune value
6 : Loop pattern
7 : Set tremolo waveform
8 : No utilizado
9 : Retrigger sample
10 : Fine volume slide up
11 : Fine volume slide down
12 : Cut sample
13 : Delay sample
14 : Delay pattern
15 : Invert samples

Recuadro 2: Comandos extendidos.

IMPLEMENTACIÓN

No hay que dejarse intimidar ante tal variedad de efectos. Muchos de ellos en la práctica no se utilizan nunca, de forma que la mayor parte de programas los ignoran, sobre todo los efectos extendidos más extraños, y algunos efectos complicados se implementan de forma simplificada.

Cuando en la rutina *PlayNote* se actualiza el canal con los datos de la nueva nota, se llama a la rutina *ProcessCommand*. Ésta se encarga de mirar el comando y actuar en consecuencia. Todos los efectos que tienen lugar una vez por nota se ejecutan en ese momento, como por ejemplo el cambio de tempo, el *sample offset*, etc. También en este momento se inicializan los parámetros de los comandos que deben ejecutarse en cada tick, como por ejemplo el vibrato o los *slides*, y se memorizan los comandos de salto como el *pattern break*.

La rutina *UpdateCommand* se llama en cada *tick*, y se encarga de ejecutar los comandos que se ejecutan en ese momento. Por último, cuando ya se han decodificado las notas de todos los canales, la rutina *ProcessJumps* se encarga de ver si hay que ejecutar algún comando de salto en la secuencia. Estos deben ejecutarse al final porque de lo contrario algunas notas no se decodificarían.

A la estructura de datos de cada canal virtual se le deben añadir varios campos. Uno es, por supuesto, el comando que se está ejecutando y su parámetro. Pero como se ha visto, en muchos casos si el parámetro es 0 debe tomarse el anterior del mismo efecto, de modo que hay que memorizar esos valores en un campo distinto para cada efecto, y también algunos valores calculados al decodificar el comando.

Las rutinas de mezcla de canales, temporización y control de efectos están en el fichero PLAY.PAS. Algunos de los comandos están simplificados, como por ejemplo el vibrato no utiliza semitonos para oscilar, y se ignora el BPM en el comando 15. Otros como el arpegio y varios efectos extendidos simplemente no están implementados. De todas formas, los efectos que se soportan son los que aparecerán en más del 95% de los casos.

La base de tiempos por defecto de los MODs es 1/50 de segundo

El comando 14 contiene efectos extendidos

EN LA PRÓXIMA ENTREGA

Ya se ha explicado la forma de reproducir un MOD con todo detalle, pero hasta ahora las rutinas de mezcla y reproducción de sonido sólo son mono y de 8 bits. En la próxima entrega se visitarán de nuevo las tarjetas de la serie Sound Blaster para programarlas en estéreo y a 16 bits en caso de que el modelo lo permita, y se explicará cómo modificar las rutinas de mezcla de canales para que mezclen en estéreo (ahora ya mezclan a 16 bits). Y si hay páginas suficientes también se explicará cómo adaptar el reproductor de MODs para utilizar la tarjeta Gravis Ultra Sound, una tarjeta de sonido que gracias a su sistema *wave table* evita tener que hacer la mezcla de canales por software. ■

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratatos en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

COMO SUSCRIBIRSE A

Suscríbase enviando este cupón por correo o fax (91) 320.60.72, o llamando al teléfono (91) 741.26.62 Horario 9 a 14 y 15:30 a 18:30 h

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año + regalo (filtro monitor) **por sólo** 10.995 ptas. ☐ Estudiantes carreras técnicas 40% Dto.: 8.250 ptas.

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....Telf.....Profesión.....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº .....

Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

☐ Domiciliación bancaria.
Señor Director del banco .....
Población .....

CODIGO CUENTA CLIENTE

ENTIDAD	OFICINA	DC	Nº CUENTA

Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro el recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L. como pago de mi suscripción a la revista SÓLO PROGRAMADORES.

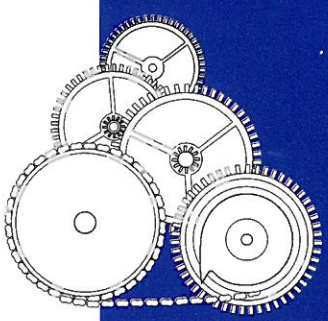
☐ Contra-reembolso del importe más gastos de envío.
☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.
☐ Giro Postal (adjunto fotocopia del resguardo).

Firma:

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Marques de Portugalete 10, Bajo

EL ARTE DE REPRESENTAR EL CONOCIMIENTO

Juan José Samper



Antes de poder hablar de representaciones del conocimiento, es necesario aclarar que se trata con dos tipos de entidades diferentes: *hechos*, que son verdades de algún mundo relevante, y *representaciones* de esos hechos en algún formalismo elegido. Para que las representaciones sean de interés en el mundo real, deben existir funciones que conviertan los hechos en representaciones y las representaciones en hechos. Para facilitar la entrada y salida de información del sistema, se necesitará también una representación en lenguaje natural, tal y como se muestra en la figura 1.

No es lo mismo un hecho que su representación

El conocimiento existe en tanto en cuanto es representable. La primera representación, de la que se parte, es una representación mental. Posteriormente se pasa a texto (hablado o escrito) en lenguaje natural. El que no exista una representación en lenguaje natural de un conocimiento no significa que no exista conocimiento.

Cuando el ingeniero tiene el conocimiento en lenguaje natural, debe enfrentarse a la tarea de representarlo de manera eficiente para que el sistema experto sea, a su vez, eficiente.

Para representar el conocimiento hay que determinar:

- El modelo o esquema de representación
- El lenguaje de representación
- El lenguaje de programación

El esquema de representación es la forma de representar el conocimiento y está muy ligado al dominio del problema que se quiere representar. No se ha encontrado un único modelo de representación, ya que el ser humano no tiene un modelo único, sino que depende de dónde lo vaya a usar. El escoger un buen modelo es muy importante.

El lenguaje de representación es el lenguaje formal en el que se representa cada uno de los correspondientes esquemas de representación de cada fragmento de conocimiento.

El lenguaje de programación es el nivel requerido al lenguaje de alto nivel que se empleará para implementar la representación dada.

No son niveles aislados, dependiendo de los esquemas de representación, vendrán los lenguajes de representación y, ligados a éstos, los lenguajes de programación.

Una primera clasificación de los modelos distingue entre representación procedural y representación declarativa.

TIPOS DE CONOCIMIENTO

El conocimiento puede ser declarativo o procedural. El conocimiento declarativo consiste en afirmaciones acerca de cosas. Con este conocimiento no se puede explicar nada, pero se pueden representar hechos y las relaciones entre ellos. Se tendrá un conjunto estático de hechos y algunos procedimientos para manipularlos. El conocimiento procedural es explicativo; da una manera de aplicar el conocimiento declarativo. Se puede indicar

La representación del conocimiento es un campo en constante investigación. No se ha encontrado una solución ideal, pero existen algunas técnicas interesantes y útiles.

la forma de ejecutar una acción. Indica qué hay que hacer y cómo hay que hacerlo.
Se distinguirá, por tanto, entre:
- Representación procedural: el conocimiento se encapsula en un algoritmo, y se trabaja con él ejecutando ese algoritmo. Son programas que expresan las relaciones entre fragmentos de conocimiento. Tiene facilidad para usar el metaconocimiento (es fácil representar cómo hacer cosas) y es indispensable a bajo nivel.
- Representación declarativa: el conocimiento no está en un algoritmo, sino que es "contado". Se tienen porciones de conocimiento independientes y fácilmente modificables. Estos conocimientos se combinan posteriormente para razonar. Las ventajas son la legibilidad, la flexibilidad y la facilidad de modificación.

El que no exista una representación en lenguaje natural no significa que no exista conocimiento

- Representación mixta: emplea una mezcla de los dos tipos anteriores.
Ejemplo: Representar la frase "Un animal nocturno con alas que no es mamífero es un ave o un insecto".
* Representación declarativa (usando lógica):
"x (nocturno(x) Ù alas(x) Ù Ø mamífero(x)
_ ave(x) Ú insecto(x))
* Representación procedural:
SI nocturno ENTONCES
SI alas ENTONCES
SI no mamífero ENTONCES
es ave O insecto

Los modelos procedurales son fácilmente empleables, pero muy rígidos (al alterar el conocimiento hay que reescribir el programa).

Para usar un modelo declarativo se necesita un motor de inferencia. La ventaja es que el conocimiento es una línea mas de declaración.

Las reglas de producción tienen tanto éxito debido a que están entre estas dos clases de representación

REPRESENTACIÓN MEDIANTE REGLAS DE PRODUCCIÓN

Este es el método de representación más común en los sistemas expertos, debido a que es un modelo bastante natural para representar el conocimiento de las relaciones causa-efecto, y a que por ser un mixto entre conocimiento declarativo y procedural, tiene las ventajas de ambos y supera las desventajas. Las reglas contienen pequeñas cantidades de conocimiento, son fáciles de entender y su uso es sencillo. Facilitan la creación y modificación de las bases de conocimientos.

Cada regla está redactada, en principio, ignorando la existencia de otras. Por ello, una regla no se activa por su nombre sino por sus condiciones. De esta manera, se pueden añadir o suprimir reglas sin preocuparse de la repercusión de las modificaciones.

El conocimiento puede ser declarativo o procedural

Una regla de producción tiene la forma:
SI antecedente ENTONCES consecuente
La parte izquierda son las condiciones de aplicabilidad y la parte derecha es la conclusión, que será una acción o una aserción (afirmación o negación) a a–adir a la base de hechos.

El antecedente y el consecuente son proposiciones (afirmaciones) o sentencias. Una proposición será una concatenación de cláusulas. Cada cláusula es una expresión que tiene tres elementos: (objeto, atributo, valor). Los objetos suelen ser los elementos que producen las causas. Un objeto se caracteriza por su nombre. Los atributos se caracterizan por un nombre y por un tipo (numérico o simbólico). Los valores pueden ser legales (los permitidos para el atributo) o específicos (los que especifican las posibles anomalías que se admiten además de los valores legales). Ejemplo, el atributo de nombre *estatura* es de tipo numérico, sus valores legales pueden estar comprendidos en el intervalo [0.35, 2.60] y sus valores específicos los formarían las condiciones o restricciones.
Un ejemplo de regla de producción podría ser:
SI (coche, matrícula, AL-5548-S)
ENTONCES (coche, es_de, Juanjo)

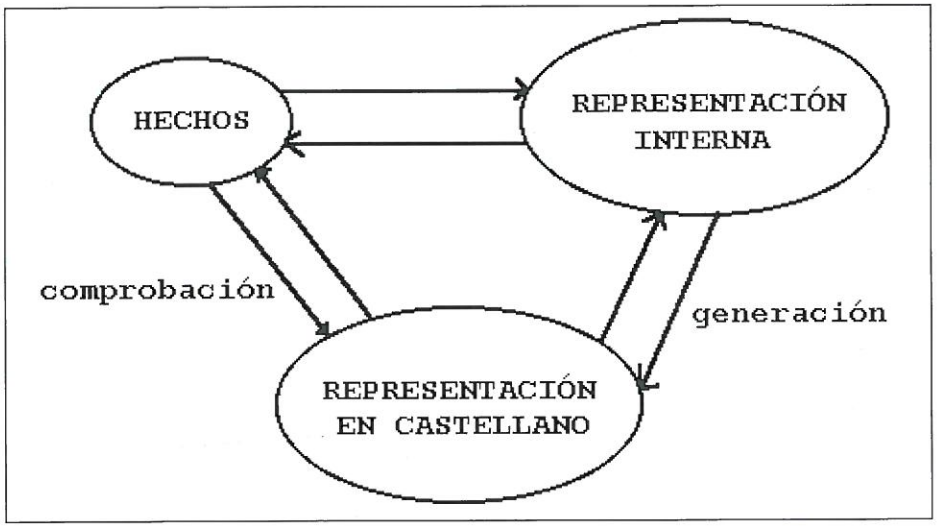
Otro ejemplo, en el que se tiene al antecedente y al consecuente de la regla en forma de afirmación, y donde se muestra el paso a la forma (objeto, atributo, valor), es:
SI tengo dinero ENTONCES me compro el coche
SI (yo, dinero, si) ENTONCES (yo, comprar, coche)

Un ingeniero del conocimiento tiene que convertir las cláusulas a ternas de este tipo. La elección debe ser unívoca en cada cláusula, pero no es única esa identificación. Habrá que elegir una representación y mantenerla a lo largo de todo el proceso.

Cuando los objetos son unitarios o unívocos, las cláusulas pueden tener la forma simplificada: (atributo, valor), pero no hay cláusulas que tengan menos de esto.

La concatenación del antecedente y del consecuente se hace mediante operadores Y/O. No se deben poner cláusulas consecuente conectadas por O, siempre tienen que unirse con Y para que todo sea verdad. Ejemplo, si se llega a una regla tal como: SI tengo dinero ENTONCES coche O barco, se deberá descomponer en:
SI tengo dinero ENTONCES coche con probabilidad 0.5
SI tengo dinero ENTONCES barco con probabilidad 0.5

Las premisas en el antecedente pueden ser tan complejas como se necesite, aunque siempre que se pueda se debe de intentar simplificar.
Las reglas se pueden utilizar de manera declarativa:
- abductivamente: SI efecto ENTONCES causa
- inductivamente: SI causa ENTONCES efecto
o de manera procedural:
SI condición ENTONCES acción



La mayoría de los sistemas expertos que utilizan reglas de producción funcionan de forma abductiva.

REPRESENTACIÓN DEL CONOCIMIENTO MEDIANTE LA LÓGICA

En este caso, cada hecho o estado se representa mediante un enunciado declarativo o proposición. Ejemplo: To–i es una mujer.

El gran filósofo griego Aristóteles desarrolló muchas de las teorías de la llamada *lógica clásica*, que trata de obtener la verdad o falsedad de una afirmación partiendo de argumentos filosóficos. Puede considerarse en muchos aspectos como una formalización del sentido común. Pero, debido a que está basada en el lenguaje natural, posee sus mismos defectos: imprecisiones y ambigüedad. Para solucionar estos problemas se ideó la *lógica simbólica* (G.W.Leibniz y posteriormente George Boole).

La lógica simbólica abstrae los conceptos mediante símbolos y representa la relación entre ellos mediante operadores especiales. Dentro de la lógica simbólica se tienen dos ramas: la *lógica de proposiciones* y la *lógica de predicados*.

La lógica de proposiciones intenta determinar la verdad o falsedad de una proposición (afirmación). Sus operadores más comunes son: Y, O, NO, _(IMPLICA) y Û(EQUIVALENTE). Se pueden crear expresiones tan complejas como se quiera combinando estos operadores. Este tipo de lógica cumple las siguientes leyes (P, Q y S son proposiciones):

- Conmutativa:
  P Y Q es equivalente a Q Y P
  P O Q es equivalente a Q O P
- Asociativa:
  (P Y Q) Y S es equivalente a P Y (Q Y S)
  (P O Q) O S es equivalente a P O (Q O S)
- Distributiva:
  P Y (Q O S) es equivalente a (P Y Q) O (P Y S)
  P O (Q Y S) es equivalente a (P O Q) Y (P O S)
- Teorema de Morgan:
  NO (P O Q) es equivalente a NO P Y NO Q
  NO (P Y Q) es equivalente a NO P O NO Q

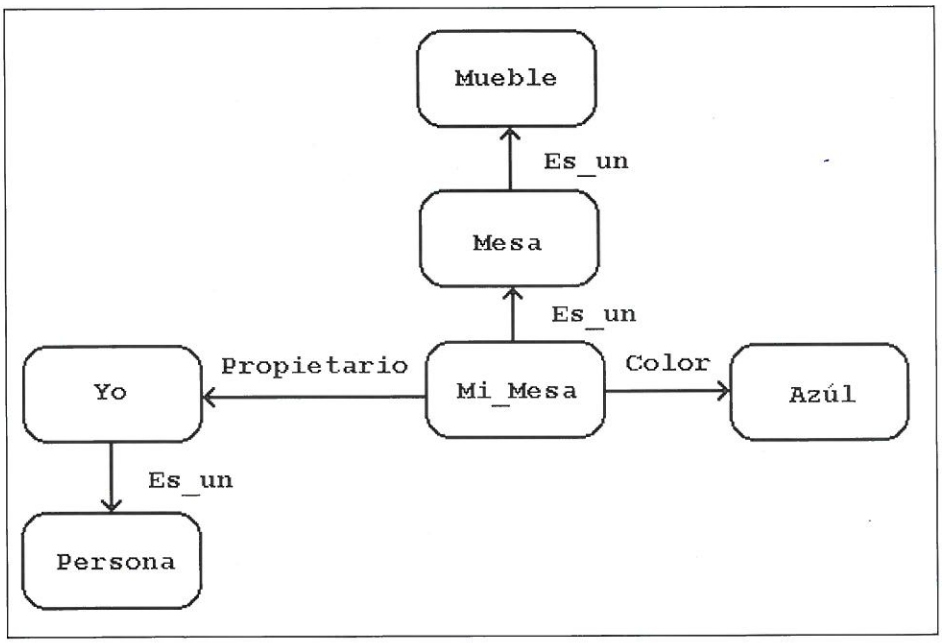
Las reglas de producción son un mixto entre conocimiento declarativo y procedural

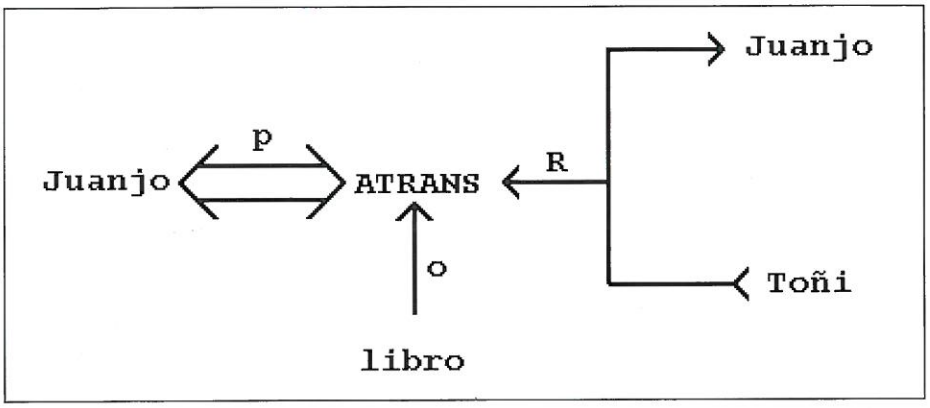
La lógica proposicional es la base de la mayoría de lenguajes de programación, tales como el C. Sin embargo, no sirve para representar todo el conocimiento. No tiene la habilidad de representar relaciones entre objetos, y sólo puede determinar la verdad o falsedad de una afirmación dada.

La lógica de predicados es una extensión de la lógica proposicional (fue una formalización de Frege en el siglo XIX). Un predicado de esta lógica se puede considerar como una función que devuelve un valor verdadero o falso. Un predicado podría ser *es_dulce*. Si con este predicado se evalúa el argumento azúcar se tendrá un resultado verdadero. Si se evalúa otro argumento como sal entonces se tendría un valor falso. La forma de expresar esto sería:

es_dulce (azúcar) ó> verdadero
es_dulce (sal) ó> falso

Utilizado lógica proposicional se podría poner:





el azúcar es dulce
la sal no es dulce

Del ejemplo, se puede observar que mientras que en la lógica de predicados se separa el objeto de su atributo, creándose una función que determina si algo es dulce o no, en la lógica de proposiciones es necesario un enunciado nuevo para cada caso.

Un predicado también puede tener varios argumentos, como por ejemplo, el predicado *más_grande* (avión, bicicleta) que dará como resultado verdadero.

El conocimiento procedural da una manera de aplicar el conocimiento declarativo.

Otra mejora importante respecto a la lógica de proposiciones es la utilización de variables, que sirve para generalizar un predicado. Ejemplo: *mujer* (x).

La utilización de cuantificadores permite resumir en una fórmula a un gran conjunto de objetos. Ejemplo: el enunciado *Todos los pájaros son aves* no se puede representar, a menos que se escriban frases separadas acerca de cada pájaro conocido. Para ello se utiliza el cuantificador ", que significa *para todo*, y el cuantificador $, que significa *existe*. Entonces, la afirmación *Todos los pájaros son aves* se expresaría:

"x pájaro(x)ó>ave(x)

y la afirmación *Hay una persona llamada To–i*:

$x persona(x) Ù nombre(x, To–i)

El símbolo Ù significa Y. Otros símbolos son Ú que significa O, y Ø que significa NO.

Muchos problemas deductivos (extraer conclusiones específicas a partir de principios generales) se expresan muy bien en este tipo de lógica, pero la mayor parte de los problemas reales que resuelve un ser humano son de naturaleza inductiva (deducir principios generales a partir de ejemplos específicos).

Para solventar algunas limitaciones de la lógica clásica, se han ideado lógicas alternativas, que se dividen en dos bloques:

- Lógicas con distintos axiomas

Se obtienen cambiando algún axioma de la lógica clásica. Las más conocidas son las lógicas multivaluadas que toman otro valor además de verdadero y falso. Una lógica de este tipo es la de Bochvar que sirve para resolver paradojas semánticas: proposiciones que no tienen valor de verdad, porque al ser verdad toman valor falso y viceversa, como por ejemplo "*esta frase es mentira*". Usará tres valores: verdadero(V), falso(F) y paradójico(P).

- Lógicas complementarias

Para obtener mayor potencia expresiva amplían el lenguaje manteniendo los axiomas básicos. Una lógica de este tipo es la *lógica situacional* que sirve para poder representar mejor los conocimientos de un mundo cambiante. Se usa para la planificación de robots. En ella todos los predicados tienen un argumento extra que denota la "situación" en la que la fórmula es cierta. Ejemplo: sobre(b1, b2, S1) afirma que b1 está sobre b2 en la situación S1.

REPRESENTACIÓN ESTRUCTURADA DEL CONOCIMIENTO

Mediante la lógica es fácil representar y manipular hechos simples, por desgracia, el mundo es más complicado y es necesario disponer de estructuras que permitan describir el conocimiento complejo y estructurado de un dominio. Las más importantes son:

- Redes semánticas
- Dependencia conceptual
- *Frames*
- Guiones

Estas estructuras comparten entre sí el que las entidades complejas se pueden describir como un conjunto de atributos y sus valores asociados.

REDES SEMÁNTICAS

Una red semántica es un grafo dirigido, en el que los nodos representan entidades (objetos o sucesos), y los arcos que los conectan representan las relaciones entre las entidades.

Tienen su origen en las ideas de asociacionismo y atomización de Aristóteles. La conducta humana cognitiva está marcada por la asociación. A su vez existe el reduccionismo o subordinación: la semántica de un concepto se obtiene como consecuencia de conceptos más generales.

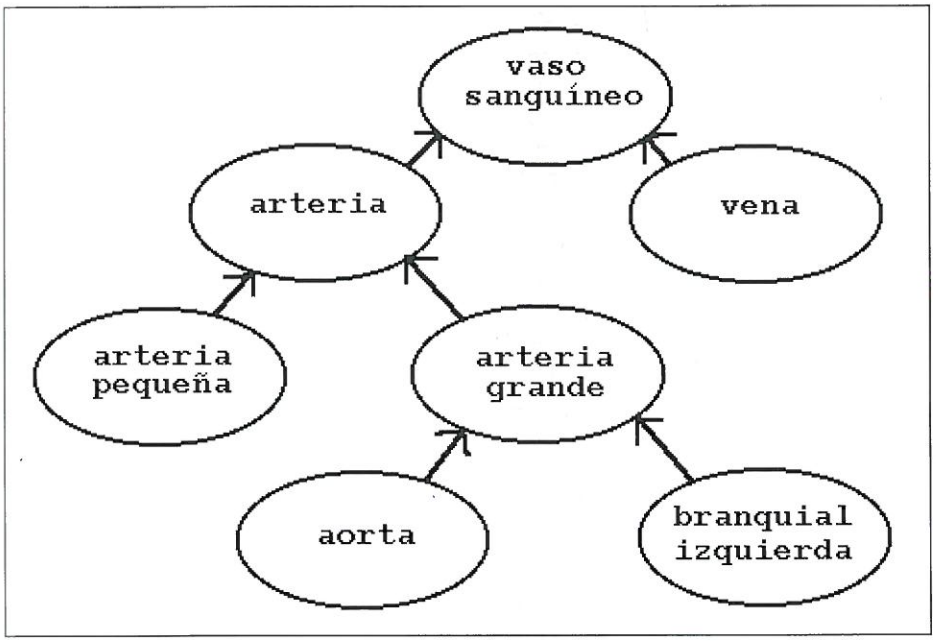
Ross Quillian construyó en 1966 el primer modelo computacional válido de red semántica. Usó arcos de la forma:

en donde A es la relación que liga a B y C. El arco se etiqueta con el nombre del tipo de relación y una entidad simple se representa por un nodo simple. Un ejemplo de red semántica se puede ver en la figura 2.

Las relaciones se establecen mediante arcos "Î" y "comentario", mientras que con "Í" se indica que es subconjunto (Es_un).

Las redes semánticas no están preparadas para hacer inferencia salvo la que le es propia, es decir, si una entidad está ligada con Î o con Í a otra entidad, hereda las características de la última. Ejemplo, en Aó>B, B heredaría las características de A. Existe una forma de hacer inferencia y es por *acoplamiento*. Ejemplo, suponiendo la red:

y la metaregla:



Acoplando los arcos se puede inducir que Pepe es nieto de Felipe.

Las redes semánticas son buenas para representar conocimiento declarativo con estructura jerárquica. Además de ser una herramienta visual, una red semántica puede programarse para formar una base de conocimientos.

DEPENDENCIA CONCEPTUAL

Es una teoría que se utiliza para representar el significado de frases del lenguaje natural.

Según R. Shank, todo lo que las personas hablan puede representarse usando un conjunto de conceptos. Las unidades básicas de significado se estructuran como subconjuntos de conceptos y se denominan *conceptualizaciones*. Éstas se enlazan para construir estructuras complejas. Los bloques de construcción básicos son:

- ACT: acciones
- PP: objetos o actores causantes de los actos
- AA: modificadores de acciones
- PA: modificadores de objetos

Ejemplo, en la frase "Paco está gordo", Paco es un PP y gordo es un PA.

Existe un conjunto de *acciones primitivas* como ATRANS que significa transferencia de algo (dar o donar) o ATTEND que es el enfoque de un órgano de los sentidos (como escuchar). A partir de ellas se construyen acciones de un nivel más elevado. Un ejemplo de representación usando dependencia conceptual se muestra en la figura 3.

Representar el conocimiento usando el modelo de dependencia conceptual facilita el razonamiento: se necesitan menos reglas de inferencia y muchas inferencias están ya contenidas en la representación. Sin embargo, tiene inconvenientes: se necesita descomponer todo el conocimiento en primitivas de bajo nivel y sólo se pueden representar acontecimientos.

El conocimiento procedural da una manera de aplicar el conocimiento declarativo

MARCOS O FRAMES

El concepto de *frame* lo introdujo Minsky en 1975. Éste estudió la mente humana fijándose en la memoria de representación, es decir, en la capacidad del ser humano para reconocer objetos por medio de estereotipos que se ha creado de ellos anteriormente. Un estereotipo es un conjunto de características generales de un objeto. Ejemplo: nadie representa mentalmente una silla concreta, sino un estereotipo de ella y, cuando se le presenta una, comprueba si se adapta o no al estereotipo del que dispone, para poder concluir si se trata de una silla. Las acciones también tienen un estereotipo, por ejemplo comer (aunque dos personas comen de forma distinta).

Cada ejemplo de objeto es una instanciación del estereotipo del objeto. Para Minsky un estereotipo representa una colección de objetos. Así un estereotipo va a ser un "casillero" o *frame* que tiene una serie de ranuras o *slots* correspondientes a los atributos específicos del objeto. Cada *slot* puede contener a su vez una o varias facetas. La faceta puede contener el valor del atributo, o un valor que se toma por defecto, o un procedimiento que se llama para obtener el valor.

Para reconocer un objeto, se busca un *frame* y se rellenan los *slots* correspondientes a las características de ese *frame*. Si se rellenan suficientes *slots*, se puede concluir que el objeto pertenece a la clase de objetos representados por el *frame*. Cada estereotipo (o *frame* vacío) representa una clase de objetos y cada objeto es un *frame* relleno.

En general, un *slot* corresponde al par (atributo, valor). Los valores que puede tener un atributo serán:

- Específicos, como por ejemplo "número de hijos".
- Genéricos, que son valores no discutibles puesto que los poseen todos los elementos de esa clase, como, por ejemplo, "sangre caliente" para un mamífero.
- Por defecto, que son valores que se pueden suponer si no se conoce otro valor, como, por ejemplo, "dos piernas" en un ser humano (se supone que es así, pero puede no tenerlas).

Cuando se analiza un estereotipo, primero se buscan los valores genéricos, después los valores por defecto y por último los valores específicos.

Un *slot* también puede contener:

- Condiciones, que restringen los posibles valores de relleno.
- Procedimientos, para determinar el valor del *slot*.
- Punteros, por ejemplo, a otro *frame*.

La estructura general de un *frame* puede tener el siguiente aspecto:

```
(nombre
  (slot 1 (faceta 1 (valor 1, ...))
          (faceta 2 (valor 1, ...))
          .
          .
  (slot 2 (faceta 1 (valor 1, ...))
          .
          .
```

Un ejemplo de *frame* es el de la figura 4. Como puede observarse, la clase *frame* tiene mucho en común con el tipo de dato registro proporcionado por el lenguaje Pascal.

Los *frames* se pueden organizar en estructuras jerárquicas. En una jerarquía, el concepto más importante es la *herencia.* Esto posibilita que, dentro de un *frame*, parte de las propiedades se especifiquen como información de referencia a otros *frames* más generales. En la figura 5 se puede ver una jerarquía de *frames.*

Cada nodo es un *frame* que hereda las características genéricas del *frame* del que cuelga. Es decir, una arteria hereda las propiedades genéricas de vaso sanguíneo. Los nodos aorta y branquial izquierda son instancias del *frame* arteria grande.

Los sistemas de *frames* se usan a menudo en sistemas de reconocimiento de patrones, tanto visuales como de conceptos abstractos. Esto implica el emparejamiento de un conjunto de valores asociados a una entidad con los valores requeridos para rellenar los *slots* del *frame.*

Usando *frames*, se pueden realizar cinco tipos de inferencia:

1. Existencia inferida, al emparejar una entidad y un *frame*, se infiere la existencia de una entidad del tipo que representa el *frame.*

2. Propiedades genéricas inferidas, cuando se empareja una entidad con un *frame*, se infiere que la entidad tiene todas las propiedades genéricas asociadas al *frame.*

3. Propiedades por defecto, cuando una entidad que se ha logrado emparejar tiene un valor desconocido para una propiedad, se puede asumir que toma el valor por defecto asociado a ese *slot* en el *frame.*

4. Reconocimiento de situaciones anormales, en un *frame* la ausencia de un valor para un *slot* o la presencia de un valor no apropiado puede significar una situación no usual.

5. Inferencia por analogía, se utiliza una declaración del tipo "X es como Y" para obtener valores de *slot* para X dando valores para Y. Ejemplo: "Carmen es como una apisonadora". El valor en el *slot* peso del *frame* vehículos es relativamente alto para apisonadora. Como no se conoce ningún valor para el peso de Carmen, se selecciona un valor alto para ese *slot*, en analogía al valor alto en apisonadora.

Los *frames* son atractivos intuitivamente, ya que las personas usan normalmente el concepto de agrupar propiedades relacionadas con una entidad o un conjunto de entidades. Sin embargo, no hay una teoría formal de *frames* y, por ello, la inferencia y verificación de consistencia no se basan en un semántica bien definida.

GUIONES

El guión es una estructura que representa una secuencia de acontecimientos en determinado contexto. Es un estereotipo de acción temporal compleja.

Un guión consta típicamente de:

- Condiciones de entrada, que deben ser satisfechas antes de que ocurran los hechos que se describen.
- Conjunto de reglas o papeles, que son las personas que intervienen en los acontecimientos.
- Conjunto de proposiciones o material, que son los objetos involucrados.
- Conjunto de escenas, cada una de las cuales representa uno de los acontecimientos que constituyen la secuencia.
- Condiciones de salida o resultados, que se obtendrán después de que la secuencia de acontecimientos haya sido completada.

Un ejemplo de guión puede verse en la figura 6.

La utilidad de los guiones es debida a que en el mundo real existen acontecimientos conectados por relaciones causales. Los acontecimientos integrantes del guión forman una cadena causal, en donde cada acontecimiento va conectado tanto a los anteriores que lo hacen posible como a los posteriores que permite que ocurran.

UNA REVISIÓN DEL SOFTWARE

En esta ocasión, se acompa–a al artículo con el programa SISTEM2.EXE que es en esencia una revisión y mejora general del ofrecido en el anterior número de la revista. El código fuente se encuentra en el fichero SISTEM2.C y está realizado en Turbo C 2.0. Como otras veces, se entrega también una pequeña base de conocimientos (FRUTAS) para que el lector pueda experimentar.

Se ha modificado el motor de inferencia para que pueda encontrar distintas soluciones que cumplen los mismos atributos, y no se limite a dar sólo una. Cuando el sistema encuentra una solución, pregunta si se desea continuar con la búsqueda.

También se ha mejorado la respuesta del programa a la pregunta ¿por qué?. En esta versión, informará también de los objetos que ya ha rechazado como solución. Para ello se ha incluido la estructura:

```
struct rechazada
 {
  char accion [40];
 } rech;
```

y el *array*:

```
struct rechazada basere [TOTAL];
```

Cada vez que se rechace una solución, se llama al procedimiento *rechaza* que la incluirá en el *array*.

Otra característica nueva es que si el programa no encuentra una solución, dará un resumen con las características que se han pedido al objeto (lo que es y lo que no es según las respuestas del usuario).

De todas formas, el motor de inferencia sigue siendo bastante básico, por lo que animo al lector a que intente implementar su propio algoritmo inteligente.

En la próxima entrega, se añadirá al sistema la facultad de tratar la incertidumbre de los conocimientos que maneja.■

PROGRAMAS RESIDENTES PARA DOS (2ª PARTE)

José Galán

En el capítulo anterior de esta serie de artículos sobre programación de TSRs se comentó, aunque muy superficialmente, la forma en que el programador debe acceder a los recursos disponibles en el ordenador para no producir conflictos con el sistema operativo o con otros programas residentes o no.

En esta entrega el lector encontrará información útil sobre temas tan importantes como la gestión de memoria, uso de la interfaz *multiplex* (INT 2Fh), uso de la VGA por residentes y su temporización exacta.

GESTIÓN DE MEMORIA CONVENCIONAL

Como todos los lectores saben la memoria convencional es aquella que ocupa los primeros 640 Kb de la memoria RAM del ordenador. Sin embargo, dicha zona no es utilizable en su totalidad ya que en ella residen elementos tales como la tabla de vectores de interrupción, área de variables de la BIOS y parte del código y las estructuras de datos que forman parte del sistema operativo.

Siempre que sea posible se debe evitar el uso de memoria convencional o, en el peor de los casos, su uso debe restringirse a un mínimo que sea lo imprescindible para que su programa residente se ejecute de forma adecuada.

Esto es porque la memoria convencional donde se ejecutan todas los programas DOS que funcionan en modo real, por lo que los programas residentes deben consumir la menor cantidad posible de memoria convencional.

Otra estrategia importante a seguir es la siguiente: si su residente va a necesitar bloques de memoria convencional, solicíteselos al sistema operativo durante el proceso de instalación del residente. Si no se procede de esta forma el TSR puede solicitar memoria al DOS cuando sea activado por el usuario, pero no se tendrá garantía alguna sobre si el DOS dispondrá de la memoria libre necesaria que le ha sido solicitada por el programa residente.

Otra cuestión importante es como devolver al DOS el bloque de memoria convencional usado por un residente cuando se desea desinstalar un TSR. Generalmente esta tarea se realiza ejecutando la instalación del programa residente a la cual se la pasa, en la línea de comandos del DOS, un modificador que indica que se desea desinstalar el programa.

La forma más sencilla para realizar esta función consiste en que el residente proporcione, mediante un vector de interrupción, una pequeña API que implemente determinadas funciones como, en este caso, la liberación de las zonas de memoria utilizadas por el TSR. Para devolver la memoria al DOS, sin embargo, no es posible utilizar la función 49h de los servicios proporcionados por INT 21h. Se debe acudir, por lo tanto, a una programación del sistema de más bajo nivel, concretamente se deben seguir las estructuras de datos que mantiene el DOS con la información sobre los bloques de memoria en uso por los programas que se están ejecutando en el sistema.

El DOS mantiene, para cada bloque de memoria en uso, un bloque de control de memoria o *Memory Control Block* (MCB). Esta estructura de datos se encuentra situada en el párrafo anterior al que marca el inicio del bloque de memoria al cual hace referen-

Sin duda alguna uno de los principales escollos durante el desarrollo de programas residentes es la gestión de los recursos del ordenador (memoria, almacenamiento, entrada/salida) de una forma limpia y ordenada que no produzca conflictos con otras aplicaciones. En este capítulo de la serie trataremos éste y otros temas de interés.

cia. La estructura de este bloque de información se puede observar en la figura número uno.

El problema más grave que se plantea, en principio, es como localizar el primer MCB ya que, los restantes, van enlazados con él. Para resolver esta situación se dispone de la función 52h del DOS, la cual están indocumentada por Microsoft, que al retornar devuelve en los registros ES:BX un puntero a una serie de listas de uso interno del sistema operativo. Justo en la palabra anterior a la apuntada por ES:BX se encuentra situado el valor que indica el párrafo del primer MCB.

Una vez que se accede al primer MCB podemos ir recorriendo la lista de MCBs hasta encontrar el que hace referencia al residente que deseamos desinstalar. Para liberar el bloque de memoria ocupado por el residente tan sólo se debe escribir una palabra con el valor cero en el campo que contiene el segmento del PSP del programa, con lo que el DOS asume que se trata de un bloque de memoria libre y que, por lo tanto, puede usar. En el listado número uno se puede observar un programa de ejemplo que recorre la lista de MCBs del DOS.

Durante el proceso de desinstalación es imprescindible, además, restaurar el contenido de los vectores de interrupción que hayan sido utilizados por el TSR, de forma que una vez desinstalado el residente no quede del mismo rastro alguno en el sistema.

USO DE LA INTERRUPCIÓN MULTIPLEX

La interrupción denominada *multiplex* se corresponde con el vector de interrupción 2Fh y, por lo tanto, es activada mediante una instrucción INT. El uso de esta interrupción se centra, casi exclusivamente, en la implementación de servicios que son usados por aplicaciones.

Un ejemplo claro es el controlador de memoria extendida, HIMEM.SYS, cuya presencia se detecta mediante esta interrupción.

Queda claro que puede resultar muy atractivo el uso de dicha interrupción para implementar, por ejemplo, residentes que dispongan de una pequeña API que sea llamada, bien por ellos mismos, o por otros programas.

Sin embargo se deben seguir ciertas precauciones a la hora de desarrollar un programa que intercepte la interrupción 2Fh. Esto es así ya que, dicho vector, es usado por multitud de programas y, absolutamente todos, deben tomar momentáneamente el control del ordenador cuando se ejecuta una instrucción INT 2Fh.

Por lo tanto el funcionamiento de un TSR que capture el vector 2Fh debe ser el siguiente: primero, activar las interrupciones, mediante la instrucción STI, para de esta forma no bloquear las interrupciones producidas por el hardware; segundo, comprobar si el programa que ha ejecutado la instrucción INT 2Fh está solicitando servicio a su residente y, si es así, proporcionar dicho servicio y devolver el control al programa que ha llamado al vector 2Fh; tercero, si no se solicita servicio a su residente debe pasar el control a la dirección de memoria apuntada por el contenido anterior del vector 2Fh.

Deben seguirse ciertas precauciones en los programas que intercepten la interrupción 2FH

Aparentemente, la cuestión más complicada es cómo identificar a un programa que requiere un servicio de uno de los muchos residentes instalados en el sistema que han capturado el vector 2Fh. Para resolver este problema existen diversas soluciones, siendo las siguientes las más comúnmente utilizadas.

La primera solución consiste en comprobar el contenido de los registros del procesador para, de esta forma, comprobar la existencia de determinados valores que indican que el programa que ha realizado la llamada está realizando una petición de servicio a un residente en concreto. De esta forma el lector puede definir unos valores determinados y su posición en el juego de registros de la CPU mediante los cuales el residente sabe cuando se le está haciendo referencia.

El segundo método se basa en pasar un puntero (en los registros DS:SI, por ejemplo) que apunte a una cadena cuyo contenido indica que se está invocando a un TSR en concreto.

TABLA 1		
Dirección	Contenido	Longitud
0	"Z"=último MCB, "M"=hay más MCBs	BYTE
1	Segmento del PSP asociado al MCB	WORD
3	Número de párrafos del bloque referenciado	WORD
5	Sin usar	11 BYTES
16	Comienzo de la zona referenciada por el MCB	Variable

Estructura de los MCB (Memory Control Blocks) del DOS.

CONSERVACIÓN DEL ESTADO DE LA VGA

Es muy posible que muchos de los residentes que los lectores piensen desarrollar realicen accesos al hardware de la VGA o que, en muchas situaciones, el programa que se estaba ejecutando en primer plano programe directamente los registros de la tarjeta gráfica.

En cualquiera de los dos casos su TSR debe guardar el estado de la VGA antes de ejecutarse y, al retornar al programa que fue interrumpido por el residente, restaurar dicho estado. Debemos advertir que los consejos que se van a dar en esta sección son aplicables a VGAs funcionando en modos estándar de IBM o en alguno de los llamados "modos X". Los modos extendidos, propios de cada *chipset* VGA, utilizan registros adicionales que no se encuentran recogidos en el estándar que en su día fue definido por IBM.

Cualquier programador familiarizado con el hardware VGA sabe que los bloques de registros más importantes de la tarjeta gráfica son los siguientes: CRTC (direcciones 3X4h, 3X5h), *timming sequencer* o TS (direcciones 3C4h y 3C5h), *Graphics Data Controller* o GDC (direcciones 3CEh y 3CFh) y *ATtribute Controller* o ATC (dirección 3C0h).

En el listado número dos se puede observar el código necesario para preservar el contenido de todos los registros de la VGA estándar.

Como es lógico, cuando su TSR vaya a devolver el control al programa que fue interrumpido, será necesario restaurar el contenido de los registros de la tarjeta gráfica. Durante esta fase es fundamental que los bloques de registros a los cuales se accede mediante un índice apunten al mismo registro a que lo hacían cuando el residente tomó el control del ordenador.

DETECCCIÓN DE LOS MODOS X DE LA VGA

Sin duda alguna el modo gráfico más atractivo de la VGA es el modo 13h, el cual proporciona una resolución de 320 por 200 pixeles con 256 colores simultáneos. El acceso al buffer de vídeo en este modo se realiza mediante el segmento A000h. Realizando unos sencillos cálculos se observa que la longitud de dicho buffer es de 64.000 bytes, por lo que tan sólo hay espacio para una página de vídeo.

Sin embargo existen varios modos, llamados modos X y que no están documentados por IBM, que hacen posible disponer de hasta cuatro páginas de vídeo y resoluciones superiores como, por ejemplo, 320 por 240 y 320 por 400 pixeles.

El secreto para conseguir este mayor número de páginas de vídeo, así como el incremento de resolución, se encuentra en la forma en que la CPU del ordenador accede a la memoria de vídeo. Desde el punto de vista del chip VGA, dicha memoria, se encuentra dividida en cuatro planos, ocupando los cuatro la misma dirección de memoria física, y seleccionándose a cual de ellos se accede mediante los dos bits bajos de la dirección de memoria en que la CPU desea leer o escribir un dato.

Gracias al hardware de la VGA, el modo anteriormente comentado, puede desactivarse. Una vez desactivado dicho modo debemos acceder a los cuatro planos utilizando un puerto de entrada/salida que selecciona el plano en el cual se va a realizar la operación de lectura o escritura. Para mayor comodidad del programador se dispone de registros independientes que seleccionan el mapa de lectura y el de escritura.

El programador de residentes tiene que consultar, únicamente, el registro de la VGA que indica si está activado el acceso lineal o mediante planos y, una vez hecho esto, acceder a la memoria de la VGA de la forma correcta dependiendo del modo activo. En el listado número tres se observa una pequeña rutina que realiza esta función.

TEMPORIZACIONES PRECISAS

Si su programa residente precisa, para determinados procesos, una temporización precisa no le quedará más remedio que utilizar el chip temporizador de que disponen los compatibles IBM PC.

Cualquier programa TSR debe almacenar el estado de la VGA antes de ejecutarse

Esto es así ya que en el mercado hay máquinas con distintos microprocesadores que corren a diferentes velocidades, no siendo difícil encontrar máquinas que siguen usando 386SX a 16 Mhz, hasta otras que disponen del potente Pentium a 120 MHz. Como se deduce fácilmente de las líneas anteriores es prácticamente imposible producir una rutina de temporización que, usando únicamente la CPU, produzca en todas las máquinas una temporización uniforme.

Como es de todos conocido, el PC dispone de un chip temporizador (consultar el artículo sobre dicho chip en el número anterior) que está dotado de tres canales o, lo que es lo mismo, tres temporizadores.

El canal cero es usado por el MS-DOS para mantener la hora y la fecha, siendo programado por el sistema operativo para que produzca una interrupción a razón de, aproximadamente, 18,2 veces por segundo. La interrupción que se produce aparece en el nivel cero del chip controlador de interrupciones y, por lo tanto, usa el vector de interrupción número ocho.

El canal uno se usaba, en el AT original de IBM, para producir los ciclos de refresco de la memoria RAM del sistema, para lo cual se usaba de forma conjunta con el canal DMA número cero.

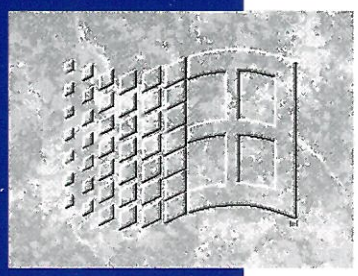
Nos resta el canal número dos del temporizador. Según IBM, la función de dicho canal es la producción de sonidos mediante el altavoz interno de que disponen todos los compatibles IBM PC. Sin embargo, si se echa un vistazo a las especificaciones del chip temporizador, se observa que es posible utilizar este canal para producir retardos de gran exactitud sin necesidad de utilizar ninguna interrupción (mediante el método denominado *polling* o sondeo), sin alterar en absoluto el funcionamiento de los canales cero y uno del chip temporizador.

En el listado número cuatro los lectores pueden observar una rutina que produce el retardo especificado en el registro AX, el cual va expresado en número de ciclos de reloj del chip temporizador (recuerde que la frecuencia de reloj del temporizador es de 1,19318 MHz).

Dicha rutina comienza activando el canal dos del temporizador, ya que dicho canal generalmente se encuentra deshabilitado mediante un bit de un registro del chip 8255. Una vez hecho esto, se programa el canal en el modo cero de cuenta, tras lo cual se escribe el valor para el contador. Para comprobar que el valor programado para el contador ha sido cargado, efectivamente, en el elemento contador se mandan comandos *lacth status* al canal dos mediante los cuales se comprueba si el valor del contador ha pasado ya a dicho elemento. Una vez en este punto tan sólo se debe esperar a que finalice la cuenta, para lo cual se envían al canal dos comandos *latch status* mediante los cuales es posible comprobar si ha finalizado la cuenta programada en un determinado canal del chip temporizador. ■

UTILIZACIÓN DE LAS CAJAS DE DIÁLOGO

Jorge R. Regidor



Existen varias maneras de introducir datos en una aplicación Windows, pero la utilización de cajas de diálogo es la más conveniente. Además, existen varios diálogos estándar definidos en Windows, de los cuales podemos hacer uso y que nos ahorrarán mucho trabajo tanto en el diseño como en la programación.

La mayoría de las aplicaciones diseñadas para Windows utilizan las cajas de diálogo para la entrada y salida de datos por parte del usuario. Este tipo de ventanas está diseñado para este uso y por eso facilita mucho al programador su tarea. El proceso de diseño del diálogo se puede hacer desde un editor de recursos, lo que simplifica mucho la tarea. Además, existen diálogos estándares para tareas específicas bajo Windows como abrir/guardar un fichero, elegir una fuente, buscar una cadena dentro de un texto, elegir un color, etc. Este tipo de diálogos son aún más sencillos de utilizar y al ser estándar permitirán que cualquier usuario de Windows esté familiarizado con ellos.

¿QUÉ SON LAS CAJAS DE DIÁLOGO?

Una caja de diálogo es una ventana hija la cual ha sido diseñada para mantener una comunicación con los usuarios. Para tal tarea, se definen una serie de ventanas hijas o controles, que se encargaran de mostrar o recibir la información. Normalmente, se abren como resultado de una elección de una opción de menú en la ventana principal, aunque esto último no tiene por que ser así.

La forma normal de utilizar un diálogo es abrirlo con unos valores por defecto para cada control, interactuar con el usuario y cerrar el diálogo guardando las modificaciones realizadas sobre variables, o estructuras del programa para realizar los cambios pertinentes.

TIPOS DE DIÁLOGOS

Existen dos tipos de cajas de diálogo en función de la forma en la cual son procesados éstos. Estos tipos son modales y amodales, aunque quizá está más difundido el nombre en inglés, *modeless*.

* Modales: Estos diálogos al ser abiertos, suspenden toda actividad de la ventana principal hasta que no sea cerrado de nuevo.

* *Modeless*: Estos diálogos continúan con toda la actividad de la ventana principal.

Se puede definir un tercer tipo, o más bien, un segundo tipo de diálogos modales como son los diálogos modales de sistema, los cuales paralizan la actividad del sistema hasta que no son cerrados. Este tipo de diálogo no conviene utilizarlo a no ser estrictamente necesario para la integridad propia del sistema.

Las cajas de diálogo son ventanas hijas especialmente diseñadas para facilitar la comunicación con el usuario

DISEÑO

Las cajas de diálogo pueden ser diseñadas en el código fuente o bien desde un fichero de recursos, si bien crear un diálogo desde programación puede lle-

gar a ser una experiencia traumática, frente a la sencillez que existe al hacerlo desde recursos. El diseño mediante estos últimos se puede hacer desde un editor de texto o desde un editor gráfico de recursos, que hoy en día es la opción a elegir por su sencillez y eficiencia. En este curso se ha ido utilizando *Resource Workshop* de Borland, por lo cual se va a indicar brevemente algunas de las opciones importantes relacionadas con los diálogos que se deberán conocer en el diseño de un diálogo desde dicho editor.

Al crear un nuevo diálogo, se pueden seleccionar las propiedades del mismo, mostrándose una ventana de propiedades como muestra la figura 1.

En esta ventana están los estilos del diálogo, de los cuales se detallan los más importantes:

* System Menu: Esta opción permite tener un menú de sistema en la esquina superior izquierda del diálogo.
* Thick Frame: Permite al diálogo extender su superficie.
* Vertical/Horizontal Scroll: Muestra las barras de desplazamiento correspondientes.
* Maximize/Minimize Box: Muestra los botones de maximizar y minimizar respectivamente.
* Absolute Align: Indica que la referencia para las coordenadas es del borde de la pantalla y no de la ventana principal.
* System Modal: Indica que el diálogo será modal de sistema.
* Visible: Indica si el diálogo dese ser mostrado después de la llamada CreateDialog, para diálogos *modeless*.

Otro punto importante a destacar son las opciones de memoria, las cuales pueden activarse pulsando la opción "Memory options" del menú "Resources". Aquí existen cuatro opciones, las cuales son:

* Load On Call: Indica que el diálogo va a ser cargado sobre memoria cuando se necesite y no al cargar el programa.
* Moveable: Indica que la memoria reservada para el diálogo no va a estar fija. Esta opción debe estar generalmente activada, ya que la memoria fija en Windows 3.1 se reserva en el área de memoria DOS, la cual suele ser escasa.
* Discardable: Esta opción le permite a Windows descargar memoria al disco duro cuando no se utilice.
* Pure: Esta opción previene que sea modificada la memoria del segmento. Si esta opción y la anterior están desactivadas la aplicación podrá modificar el diálogo.

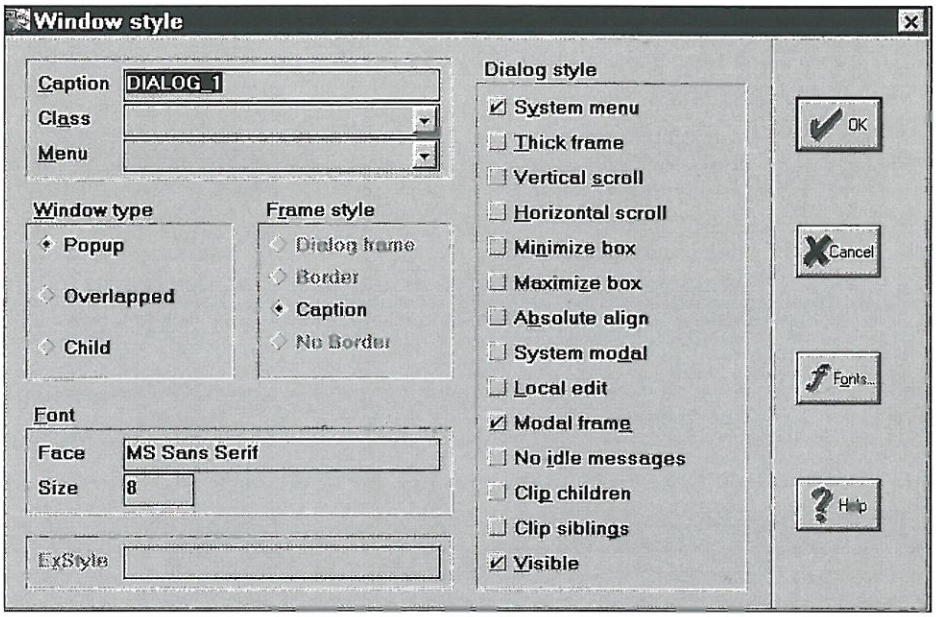


Figura 1. Propiedades de las cajas de diálogo.

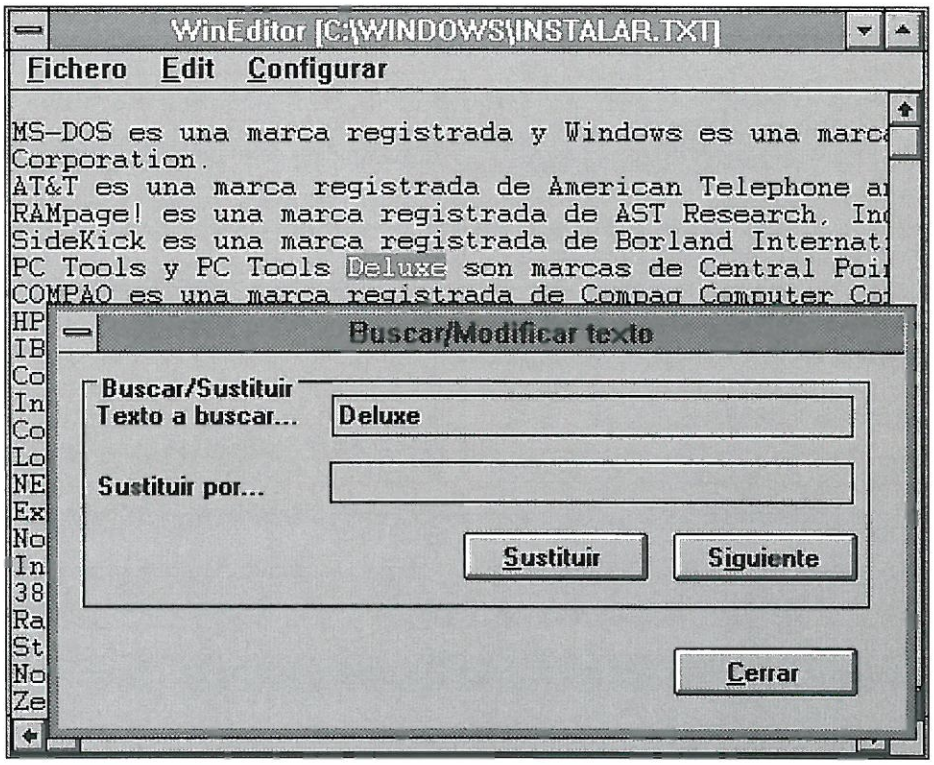

WinEditor [C:\WINDOWS\INSTALAR.TXT]
Fichero Edit Configurar
MS-DOS es una marca registrada y Windows es una marca
Corporation.
AT&T es una marca registrada de American Telephone a
RAMpage! es una marca registrada de AST Research, In
SideKick es una marca registrada de Borland Internat
PC Tools y PC Tools Deluxe son marcas de Central Poi
COMPAQ es una marca registrada de Compaq Computer Co
Buscar/Modificar texto
Buscar/Sustituir
Texto a buscar...
Deluxe
Sustituir por...
Sustituir
Siguiente
Cerrar

Figura 3. Diálogo estándar de Windows 3.1 para abrir ficheros.

El proceso de creación, de añadir controles, probar y manejo en general de Workshop no se detalla por haber sido objeto de un articulo anterior de este curso.

PROGRAMACIÓN

Antes de entrar en materia sobre la programación, se recomienda al lector que tenga presente el código fuente para contrastar lo aquí expuesto con lo mostrado por el código.

La programación de las cajas de diálogo pasa por crear una función para ges-

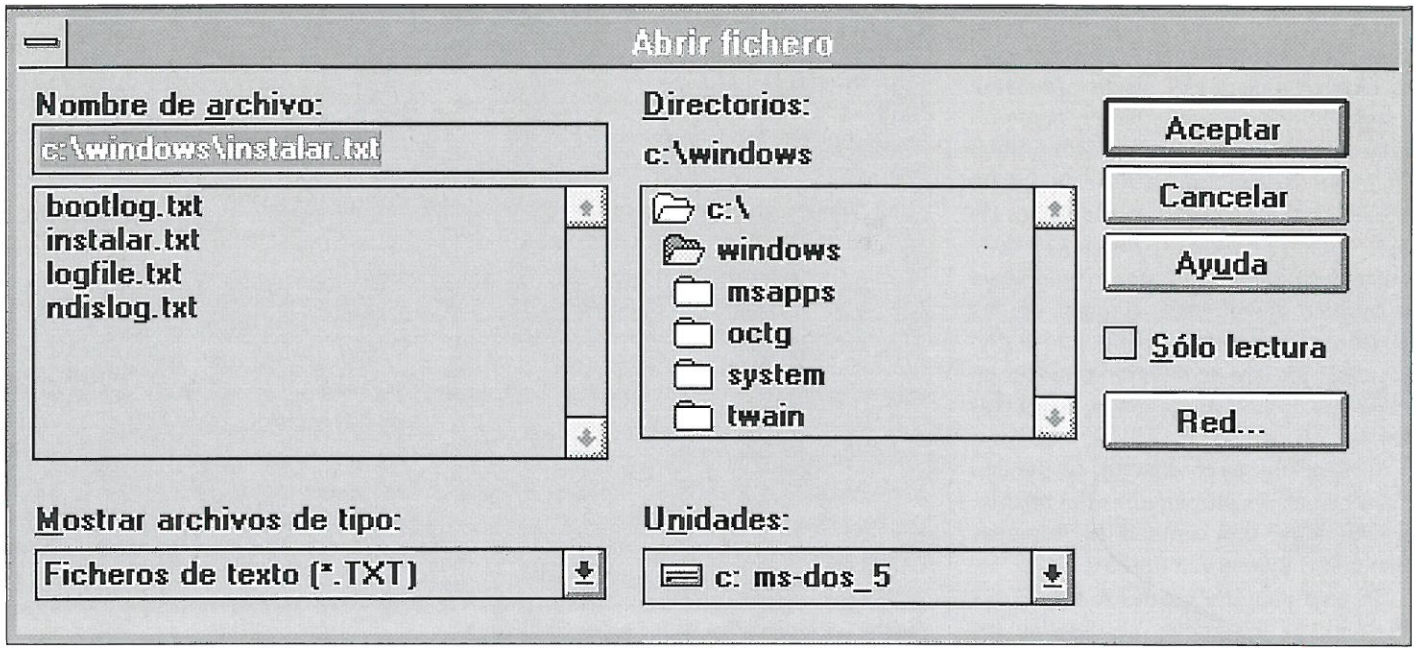


Figura 3. Diálogo estándar de Windows 3.1 para abrir ficheros.

tionar los mensajes que sean recibidos por el diálogo. El formato general de estas funciones debe ser el siguiente:

BOOL FAR PASCAL <Función> (HWND hWnd, UINT uMensaje, WPARAM wParam, LPARAM lParam)

donde: <Función>: Es el nombre que deseemos darle.

hWnd: Es el *handle* al diálogo.

uMensaje: Es el número del mensaje recibido.

wParam: Es el parámetro WPARAM del mensaje.

lParam: Es el parámetro LPARAM del mensaje.

Esta función debe devolver TRUE si el mensaje recibido ha sido atendido y FALSE, si por el contrario, no lo ha sido.

Otra importante consideración a tener en cuenta, es que está función tiene que ser exportada para que sea llamada por Windows. Para ello añadiremos su nombre en la sección EXPORTS del fichero de definiciones de la aplicación, al igual que se hace con la función de gestión de mensajes de la ventana principal.

Hasta este punto, la programación es idéntica tanto para diálogos modales como *modeless*. En este punto se va a hacer una distinción entre los diálogos modales y *modeless*.

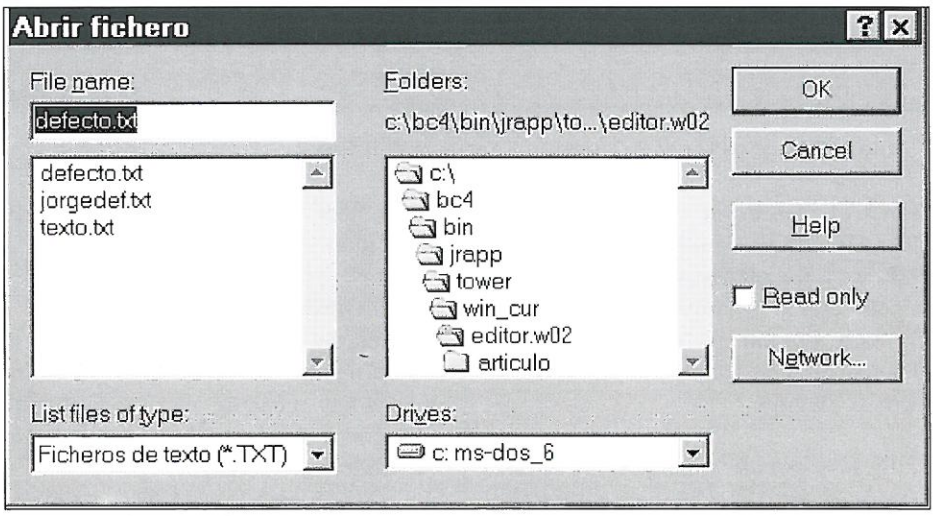


Figura 4. Diálogo estándar de Windows 95 para abrir ficheros.

CREACIÓN DE UN DIÁLOGO MODAL

La forma usual de crear un diálogo es como resultado de una opción de un menú, aunque no tiene porqué ser así. La función para crear el diálogo es *DialogBox* y esta función no retorna el control hasta que el diálogo no es cerrado. El proceso típico para crear un diálogo es el siguiente:

```
dlgSetupProc = MakeProcInstance((FARPROC)wndProcSetupDlg, hInst);
DialogBox(hInst, "SETUP_DIALOG", hWnd, dlgSetupProc);
FreeProcInstance(dlgSetupProc);
```

donde: dlgSetupProc es una variable de tipo DLGPROC.
wndProcSetupDlg es el nombre de la función de gestión de mensajes.
hInst es el *handle* a la instancia de la aplicación.
"SETUP_DIALOG" es el nombre en el fichero de recursos del diálogo.

La función *MakeProcInstance* se encarga de unir la función dada con el segmento de datos de la instancia, para que la dirección devuelta pueda ser utilizada por Windows. La función *FreeProcInstance* libera la acción de *MakeProcInstance*. cualquier referencia a los datos de esta función des-

Valor	Significado
MB_ABORTRETRYIGNORE	CONTIENE TRES BOTONES: ABORT, RETRY Y CANCEL.
MB_APPLMODAL	TIPO MODAL
MB_DEFBUTTON1	EL BOTON UNO ESTÁ ACTIVADO POR DEFECTO
MB_DEFBUTTON2	EL BOTON DOS ESTÁ ACTIVADO POR DEFECTO
MB_DEFBUTTON3	EL BOTON TRES ESTÁ ACTIVADO POR DEFECTO
MB_ICONASTERISK	MUESTRA EL ICONO ASTERISCO
MB_ICONEXCLAMATION	MUESTRA EL ICONO DE EXCLAMACION
MB_ICONHAND	MUESTRA EL ICONO DE STOP
MB_ICONINFORMATION	MUESTRA EL ICONO DE INFORMACION
MB_ICONQUESTION	MUESTRA EL ICONO DE PREGUNTA
MB_ICONSTOP	MUESTRA EL ICONO DE STOP
MB_OK	CONTIENE UN SOLO BOTON, OK
MB_OKCANCEL	CONTIENE DOS BOTONES: OK Y CANCEL
MB_RETRYCANCEL	CONTIENE DDOS BOTONES, RETRY Y CANCEL
MB_SYSTEMMODAL	TIPO MODAL DE SISTEMA
MB_TASKMODAL	IGUAL QUE MODAL PERO DESACTIVA LA APLICACION ACTIVA
MB_YESNO	CONTIENE DOS BOTONES: YES Y NO
MB_YESNOCANCEL	CONTIENE TRES BOTONES, YES, NO Y CANCEL

tabla 1. Opciones para definir el estilo de una *MessageBox.*

pués de la llamada a esta función puede conllevar resultados fatales.

Es posible que si se utiliza un compilador reciente no sea necesario llamar a las dos funciones anteriores, por lo que la llamada quedaría en:

DialogBox(hInst, "SETUP_DIALOG", hWnd, wndProcSetupDlg);

De hecho en el API de 32bits para sistemas como Windows NT y Windows 95, estas dos funciones han quedado obsoletas.

Debe quedar claro que la función *DialogBox* no devuelve el control hasta que el usuario no cierre el diálogo. Está función retorna TRUE o FALSE producto del valor que le hallamos pasado a la función EndDialog encargada de cerrar diálogos modales. El formato de esta función es el siguiente:

EndDialog(HWND hWnd, BOOL res);

donde: HWND es el *handle* del diálogo.
res es el valor de retorno para DialogBox. Este valor se suele usar para actualizar o no, las modificaciones del diálogo sobre la aplicación.

Desde las propiedades de cajas de diálogo en Resource Workshop se puede definir todas las propiedades de un diálogo

CREACIÓN DE UN DIÁLOGO MODELESS

La apariencia de este tipo de diálogo es idéntica a un diálogo modal, salvo que nos permite continuar tareas de la aplicación. Esto queda ilustrado en la Figura 2, donde se muestra el diálogo de búsqueda de cadenas y se observa la aplicación activa.

La forma de crear una caja de diálogo *modeless* es mediante la función *CreateDialog*. Esta función crea el diálogo y retorna el control al

Valor	Significado
IDABORT	SE HA PULSADO EL BOTON ABORT
IDCANCEL	SE HA PULSADO EL BOTON CANCEL
IDIGNORE	SE HA PULSADO EL BOTON IGNORE
IDNO	SE HA PULSADO EL BOTON NO
IDOK	SE HA PULSADO EL BOTON OK
IDRETRY	SE HA PULSADO EL BOTON RETRY
IDYES	SE HA PULSADO EL BOTON YES

Tabla 2. Valores de retorno de una *MessageBox*.

programa. El formato de esta función es:

```
hSerachDlg = CreateDialog(hInst,
"SEARCH_DIALOG",
hWnd, wndProcSearchDlg);
```

donde hSearchDlg es el *handle* del diálogo y los parámetros son igual que en DialogBox.

Hay que resaltar que si en las propiedades del diálogo no se ha incluido que sea VISIBLE, después de *CreateDialog* habrá que llamar a la función *ShowWindow* para mostrarla en pantalla.

En el código fuente no se ha utilizado *MakeProcInstance* ni *FreeProcInstance* con este diálogo, para ilustrar el resultado también correcto de su no utilización. Si se utilizan estas dos funciones hay que tener especial cuidado en no llamar *FreeProcInstance* hasta que el diálogo no haya sido cerrado. La forma de cerrar el diálogo es llamando a las siguientes sentencias:

```
DestroyWindow(hSearchDlg);
hSerachDlg = 0;
```

Como se puede observar la forma de destruir la ventana es mediante la misma función que para la ventana principal. El *handle* hay que ponerlo a cero para evitar que se le continúen pasando mensajes después de su destrucción.

La forma de gestionar los mensajes es el punto de diferencia mayor entre los diálogos modales y *modeless* en cuanto a programación se refiere, y es donde se implementa la capacidad de operar simultáneamente con la ventana principal. El código encargado de tal evento es el siguiente.

```
while(GetMessage(&msgMensaje,
NULL, NULL, NULL))
{
      if(hSearchDlg == 0 ||
!IsDialogMessage(hSearchDlg,
&msgMensaje))
 {
  TranslateMessage(&msgMensaje);
  DispatchMessage (&msgMensaje);
 }
}
```

La sentencia añadida se encarga de preguntar si está el diálogo activo mediante la validez del *handle*, es decir, que sea diferente de cero. También llama a la función *IsDialogMessage*, que lo que hace es pasar el mensaje a la función de gestión de mensajes. Si el mensaje enviado es procesado por la función, y partiendo de que está bien implementada, debe devolver TRUE si el mensaje es procesado y FALSE si no lo es. Este valor de retorno es devuelto por la función *IsDialogMessage*, por tanto al ser negado implica que si el mensaje es procesado, no se vuelve a enviar a la función de la ventana principal, pero si no es procesado se envía a dicha función.

Hay que tener cuidado de evitar que el *handle* al diálogo permanezca correcto durante toda la vida del diálogo, lo cual podría dar origen a errores importantes.

DIÁLOGOS ESTÁNDAR DE WINDOWS

Existen varias cajas de diálogo estándar definidas en Windows, pero quizá la más utilizada y sencilla sea la que se crea al llamar a la función *MessageBox*, su formato es el siguiente:

```
Res = MessageBox(HWND hWnd,
LPCSTR sText,
LPCSTR sTitle, UINT fuStyle);
```

donde hWnd es el *handle* de la ventana madre.
sText es el mensaje en si.
sTitle es el título del mensaje.
fuStyle son banderas que definen los diferentes tipos de mensajes. estos estilos son los definidos en la tabla 1 y pueden combinarse mediante la utilización del operador |.
Res es un entero que contiene un valor que indica el botón que ha sido pulsado. Los posibles valores de retorno son los que se muestran en la tabla 2.

Windows define unos métodos estándar de entrada/salida de datos para unas tareas específicas, como son las

gestión de ficheros, elegir un tipo de letra, elegir un color, imprimir, búsqueda y sustitución de cadenas de texto... Todas estas tareas tienen definidos sus propios diálogos, por lo que no se tendrá que preocupar de pasarles los datos correctos para su creación, definir las opciones y de procesar los datos devueltos. Todos estos diálogos están incluidos en una librería de enlace dinámico perteneciente al sistema como es COMMDLG.DLL.

Detallar aquí todos y cada una de estos métodos sería imposible, pero existe una muy buena referencia de su utilización en la ayuda en línea del API. A modo de ejemplo, se va a tratar el diálogo encargado para abrir ficheros. Para trabajar con estos diálogos se debe incluir el fichero de cabecera COMMDLG.H. Para abrir el diálogo hay que utilizar la función siguiente:

*BOOL GetOpenFileName(OPENFILENAME FAR * OpenFileName);*

Esta función recibe una estructura del tipo *OPENFILENAME*, la cual contiene toda la información necesaria para definir y crear el diálogo, contiene información de los filtros de extensiones, el nombre del fichero devuelto. La función devuelve TRUE si el fichero ha sido localizado con éxito. El campo *lpstrFile* contiene el path del fichero, el cual nos permitirá abrir el fichero correspondiente. El diseño de este diálogo para Windows 3.1 es el que muestra la figura 3.

La función encargada de mostrar el diálogo para guardar ficheros es *GetSaveFileName*, para elegir una fuente es *ChooseFont*, para elegir un color es *ChooseColor*, para buscar un texto es *FindText*, para sustituir cadenas de texto es *ReplaceText* y para imprimir es *PrintDlg*.

Una de las ventajas de utilizar los diálogos estándar pasa como es obvio por la reducción del código a introducir, pero la otra no menos importante es la familiaridad que se le ofrece al usuario con estos diálogos al ser utilizados por el propio sistema y muchas otras aplicaciones. Una tercera ventaja es que nuestra aplicación se beneficiará de las nuevas versiones de estos diálogos en nuevos sistemas como Windows 95. Para ilustrar esto la figura 4 ilustra la caja de diálogos para abrir ficheros programada en Windows 3.1 en WinEditor pero ejecutada sobre Windows 95. Como se observa no se parece mucho a la misma versión de Windows 3.1.

Las cajas de diálogo estándar de Windows nos evitan mucho trabajo de diseño y programación

WINEDITOR VERSIÓN 0.2

Con esta segunda entrega de nuestro particular editor, se pretende ilustrar lo antes expuesto sobre la programación de cajas de diálogo. Para ello se han creado tres diálogos, uno *modeless* para buscar/sustituir cadenas de texto y dos modales, de los cuales uno es para cambiar el *password* y el otro para configurar los valores por defecto para algunas opciones del editor. Como se puede apreciar la funcionalidad de estos dos últimos diálogos no está implementada totalmente, ya que la resolución final será fruto de la nueva entrega de nuestro curso.

El esqueleto fundamental de un diálogo es la función de gestión de mensajes, muy similar a la función de gestión de mensajes para la ventana principal

Como se puede observar, el código fuente ha ido creciendo con la nuevas implementaciones, pero lo que realmente importa en esta entrega es la programación en cuanto a cajas de diálogo se refiere, por tanto no hay que perderse en intentar entender todo el código sino de extraer la información importante para nosotros a día de hoy, y esperar a que en la próxima entrega se solventen algunos otros puntos. Esta situación se da por las necesidades propias de crear un esqueleto un programa con una funcionalidad mínima.

Se Insta al lector a que se fije especialmente en la comunicación entre el diálogo y sus controles mediante funciones como *GetDlgItem* para obtener el *handle* del control a partir de su identificador, *GetDlgItemText* para obtener el texto de un control, *CheckDlgButton* para poner el estado de un *checkbox* o *radiobutton*, *IsButtonChecked* para saber el estado de estos últimos.

CONCLUSIÓN

Las cajas de diálogo nos permiten realizar la comunicación entre la aplicación y el usuario, por lo cual forman parte importante en la programación Windows. Se han expuesto los tipos de diálogos que se pueden crear y se ha hablado de los diálogos estándar, incidiendo en su utilización siempre que se pueda.

Por otra parte, se ha resaltado la diferencia del modo de operar entre un diálogo modal, un *modeless* y un modal de sistema, el cual debe restringir su uso a situaciones críticas para el propio sistema.

SIGUIENTE ENTREGA

Se pretenderá crear un fichero de configuración del sistema para almacenar las configuraciones por defecto, así como para almacenar el *password*. Se tratará el tema de acceso, lectura y escritura en ficheros; así como de las funciones encargadas de acceder y modificar los ficheros de configuración. Para ello, se creará un fichero WINEDIT.INI. con todos los datos del programa. ■

COMPILACIÓN DEL NÚCLEO DE LINUX

David Aparicio

Para aprovechar las características de nuestro hardware, o bien para incorporar modificaciones, es imprescindible conocer cómo se puede reconstruir el núcleo de Linux. Aquí explicaremos la forma de realizar esta tarea.

Un aspecto vital de Linux consiste en la disponibilidad de las fuentes de cualquier programa ejecutable en nuestro PC. El "programa" por excelencia en nuestro sistema es, precisamente el núcleo. Entendemos como tal el binario que se carga en los primeros instantes de encender nuestro PC, y que permanecerá ejecutándose hasta que apaguemos, gobernando así todos los dispositivos de nuestra máquina y ofreciendo al resto de programas servicios que deseemos ejecutar.

En la instalación de Linux, se graban en nuestro disco duro todos los ficheros fuentes que pueden componer dicho núcleo, a partir del directorio */usr/src/linux*. Además de este árbol de fuentes, podemos observar un fichero que se encuentra en el directorio raíz, denominado *vmlinuz*, que es la imagen compilada que se carga en nuestra máquina al arrancar.

Este fichero, creado en la instalación del sistema, puede no ajustarse completamente a nuestras necesidades. Bien por falta o por exceso de *drivers* (los programas que manejan cada periférico de nuestro PC), nuestro núcleo puede no aprovechar todo lo que tenemos instalado, o bien ocupar memoria inútilmente, con características que nunca se utilizan en nuestra máquina. Al concluir este artículo, habremos producido un fichero imagen que sustituirá al anterior, y mejorará todos estos aspectos.

POR QUÉ SUSTITUIR EL NÚCLEO

La mayor parte de "nuevas características" o "corrección de fallos" afectan al núcleo de Linux. Por ello, aparecen versiones nuevas con más frecuencia que del resto de programas.

Cuando se considera que ha habido un cambio global importante, se lanza una "distribución", es decir, una versión de núcleo Linux junto con las versiones más actualizadas de programas y utilidades asociados, ocupando en total del orden de cientos de megabytes, y sería muy costoso (y pesado) reemplazar todo cada vez. Por ejemplo, con la distribución Slackware 2.1 se ofrecía la versión del núcleo 1.1.59, y en la 2.2 se incluyó la versión de núcleo 1.2.3.

Sin embargo, para solucionar pequeños problemas o incluir *drivers* de dispositivos novedosos, se lanza con mayor frecuencia una nueva versión del núcleo, la cual sólo ocupa entre dos y tres megabytes. En el momento de distribuir el CDROM de Sólo Programadores con la distribución 2.2, la versión de núcleo más moderna (la cual actualiza a la "oficial", 1.2.3) era la 1.2.8. Los lectores pueden encontrar el archivo que contiene este núcleo, en el directorio del CD */lib/lnx128.tgz*.

Aunque podemos descomprimir el nuevo núcleo encima del árbol antiguo, se recomienda no seguir esta práctica para no dejar posibles ficheros obsoletos ocupando espacio inútil. Los comandos que siguen nos permiten mantener la estructura antigua en un árbol, y descomprime un nuevo núcleo (supondremos la versión 1.2.8):

```
cd /usr/src
mv linux linux.old
tar xvfz /mi/path/lnx128.tgz
```

A partir de este momento, sólo nos quedaría elegir la configuración que deseemos, compilar e instalar el nuevo núcleo, como veremos en los próximos

apartados. Si deseamos borrar el viejo árbol, teclearemos:

cd /usr/src
rm -r linux.old

Antes de continuar, comentaremos otra forma de actualizar dicha estructura de directorios.

Aunque el archivo que contiene el nuevo núcleo apenas ocupa unos megas, esto sigue siendo excesivo para una pequeña actualización de código. En realidad, todo tamaño superior a un disquete debiera considerarse "excesivo" para ciertas aplicaciones. Por ello, se suelen distribuir también ficheros muy pequeños, de menos de 300 kilobytes, con un resumen de las diferencias que se introducen de una versión a otra.

Es decir, si disponemos del archivo que contiene el núcleo 1.2.8, para actualizarnos al 1.2.9 no necesitamos volver a obtener el nuevo archivo completo. Bastaría un pequeño fichero con el resumen de los parches a introducir, ocupando un espacio quizá centenas de veces menor. En el disco que acompaña a la revista, incluimos, por ejemplo, dicha actualización, en el archivo *patch129.gz*. Para aplicarla, podemos seguir los siguientes pasos:

```
cp /mi/path/para/patch129.gz /usr/src
cd /usr/src
gzip -d patch129.gz
patch -p0 <patch129
```

Los siguientes comandos deberían producir (siempre suponiendo que en */usr/src/linux* tenemos instalados los fuentes de la versión 1.2.8) la actualización del árbol de fuentes a la versión 1.2.9. A partir de ahora, podremos proceder a configurar el nuevo núcleo.

ELIGIENDO DISPOSITIVOS

Una vez que tenemos el árbol adecuado, nos situamos en el directorio */usr/src/linux*. En todo el proceso que va a seguir, emplearemos la utilidad *make*, y el compilador de C. Por supuesto, debemos tener ambos instalados en nuestro sistema antes de proseguir. Todo esto lo encontraremos en la serie D.

¡Makei es un programa que permite automatizar la ejecución de una serie de comandos en base a unas reglas determinadas, produciendo un método óptimo para compilar grandes aplicaciones, donde se involucran muchos ficheros fuente y objetos.

En primer lugar, podemos inicializar todo el árbol con:

make mrproper

Este simpático comando limpia (como podemos deducir por el alarde publicitario) ficheros intermedios que pudieran quedar de anteriores configuraciones. Suele no ser necesario invocarlo, pero es interesante conocer su existencia por si se necesitase.

Ahora procedemos a elegir opciones que se desean incluir o eliminar del núcleo. Se suele responder con un sólo carácter, `y´ para incluir una opción y íní para eliminarla. Observe la figura 1 para tener una visión global de las opciones disponibles. A continuación vamos a examinar las mas interesantes. Invoque la configuración con:

make config

A continuación, realizaremos una descripción breve sobre las preguntas mas interesantes:

- ***Kernel math emulation:*** Habilita la emulación de coprocesador, para 386SX, 386DX, 486SX y Pentium con bug. Sin embargo, si se detecta un coprocesador correcto, se emplea el hardware y deshabilita la emulación, en el arranque.
- ***Normal (MFM/RLL) disk and IDE disk/cdrom support:*** Se debe habilitar siempre, excepto si tenemos un sistema que utiliza solamente SCSI. En caso de responder afirmativamente, aparecen cuestiones adicionales:
- ***Old disk-only driver for primary i/f:*** Habilita el *driver* de discos MFM e IDE para dos unidades de discos. Responda negativamente si tiene una controladora secundaria de discos, o una EIDE.
- ***Use new IDE driver for primary/secondary i/f:*** Habilitar el nuevo *driver* IDE en Linux. Imprescindible si tiene cdrom IDE. El antiguo está quedando en desuso, y sólo debería responder no si tiene problemas con los nuevos *drivers*.
- ***Include new IDE driver for secondary i/f support:*** Si ha seleccionado el manejador antiguo para las unidades primarias, se puede todavía cargar el nuevo manejador para las secundarias. Esto habilita la opción de cuatro discos duros/cdrom IDE.
- ***Include support for IDE/ATAPI CDROMs:*** Si dispone cdrom de tipo IDE/ATAPI (los que se enganchan a la misma controladora IDE/EIDE que su disco duro), este es el *driver* a habilitar para que funcionen.
- ***XT harddisk support:*** Sólo para discos duros "históricos" de XT con controladoras de 8-bit. Normalmente se responde que no.
- ***Networking support:*** Responda sí si va a utilizar cualquier tipo de enganche a red, tanto IPX, TCP/IP o protocolos de línea serie (SLIP o PPP).
- ***Limit memory to low 16Mb:*** Si va a utilizar pocas aplicaciones, siempre en los primeros 16 megas de memoria, responda sí. Esto suele optimizar el rendimiento del sistema. Elija lo que le convenga según la RAM que tenga instalada.
- ***PCI bios support:*** Si tiene periféricos PCI, responda afirmativamente.
- ***System V IPC:*** Habilita comportamientos específicos de gestión de memoria y semáforos, al estilo System V. Imprescindible si va a ejecutar emulaciones de SCO o DOS.
- ***Kernel support for ELF binaries:*** Soporte para ejecutar programas ELF de forma nativa en Linux. Este formato es un estándar en las versiones x86/Unix, y se acabará tomando en un futuro cercano como el ejecutable nativo de Linux. De momento, es útil sólo si va a emular SCO. Nótese que esto solamente capacita para la ejecución de programas ya compilados. La construcción de nuevos ejecutables con este formato requiere un compilador de C con esta capacidad.
- ***Set version information on all symbols for modules:*** Los módulos son la capacidad que está adquiriendo el núcleo de Linux para cargar los *drivers* de los dispositivos que se utilicen en el sistema, optimizando el uso de memoria y el rendimiento general. Esto tiene mucho sentido si usted está compilan-

do un núcleo para instalarlo al mismo tiempo en varias máquinas, puesto que permite compilar un núcleo con las partes comunes, y luego añadir los módulos específicos de cada máquina después del arranque. Si responde sí a la incorporación de un manejador determinado en las preguntas siguientes, se incluirá en el núcleo principal. Si responde que no, se construirá un módulo para cargarlo por separado (con *"make modules"*), tras el arranque. Actualmente la lista de *drivers* "modularizables" no es muy grande, por lo que se recomienda usar los módulos sólo para usuarios curiosos que deseen experimentar esta nueva opción. La lista de *drivers* "modularizables", e instrucciones de uso, se encuentra en *"/usr/src/linux/README.modules"*.

- ***TCP/IP networking:*** Si eligió tener capacidades de red, responder sí ahora es casi obligado. El protocolo TCP/IP es el más extendido en la actualidad, y se usa siempre en estaciones Unix. En el futuro, será posible no elegir este protocolo, si usted desea integrar Linux sólo en redes Novell, AppleNet o AX25.

- ***IP forwarding/gatewaying:*** Sólo responda afirmativamente si va a utilizar Linux como enrutador entre dos redes ethernet, o como servidor PPP o SLIP. Esta opción capacita a un nodo con Linux para leer paquetes de los que no es destinatario, cuando deben transportarse de una red a otra.

- ***IP multicasting:*** Esta opción está en experimentación, y habilita al núcleo TCP/IP para enviar paquetes con varios destinatarios simultáneos. La diferencia entre *multicast* y *broadcast* es que, en el primer caso, se puede precisar el destino de los mensajes. Esta opción requiere un software de comunicaciones preparado para explotar dicha característica. Normalmente se responderá que no.

- ***IP firewalling:*** Un nodo *firewall* aisla una red de ordenadores del mundo exterior, dificultando el ataque al sistema por parte de personas malintencionadas, protegiendo ciertos aspectos de las comunicaciones, y aumentando la seguridad general de la red. Requiere software especializado adicional. Normalmente se responderá que no.

- ***IP accounting:*** Permite realizar mediciones de los destinos y servicios empleados en las comunicaciones, para realizar funciones de auditoría. Interesante para servidores de servicios InterNet. Requiere software adicional. Normalmente se responderá no a esta opción.

- ***PC/TCP compatibility mode:*** Compatibilidad con redes PC/TCP, debido a un bug en el software de estos sistemas. Normalmente se responde no.

- ***Reverse ARP:*** El protocolo RARP permite asignar direcciones IP en el arranque, iniciando la carga por red de máquinas que no disponen de disco, como terminales X. Normalmente se responderá que no.

- ***Assume subnets are local:*** Simplificación en el núcleo TCP/IP de Linux. Todavía no está completamente probado el deshabilitarlo. Responder sí.

- ***Disable NAGLE algorithm:*** El algoritmo NAGLE intenta mejorar el rendimiento de la red IP para mensajes pequeños, agrupándolos en base a temporizaciones antes de enviarlos. Normalmente está habilitado. Responder no.

- ***The IPX protocol:*** Como se ha comentado antes, la implementación de este protocolo (todavía sin codificar completamente) permitirá integrar máquinas Linux en redes Novell, actuando como servidores, o bien para hacer puentes entre redes Novell y TCP/IP. Por el momento, sólo es útil si va a utilizar el emulador DOS.

- ***SCSI support:*** Responda sí, si su sistema dispone de algún controlador SCSI. Es posible tener simultáneamente soporte IDE y SCSI. Si responde afirmativamente, aparecerán preguntas adicionales para habilitar los periféricos deseados. No vamos a comentar aquí toda la lista de cuestiones. Acuda al gráfico 1 para ver los *drivers* disponibles.

- ***Network device support:*** Tras elegir los dispositivos SCSI (si ha sido el caso), pasaremos aquí a elegir el manejador apropiado para nuestra tarjeta de red. A menos que vaya a experimentar con la opción de módulos cargables en ejecución, responda sí.

- ***Dummy net driver support:*** El dispositivo *dummy* es un truco que se emplea en sistemas SLIP (TCP en líneas serie) para asignar direcciones IP provisionales. Si no le suena nada de todo esto, no se preocupe, y elija ínoí para ésta y las dos siguientes cuestiones.

- ***SLIP (serial line) support:*** Responda sí, si va a utilizar el protocolo SLIP de conexión IP en líneas serie.

- ***PPP (point-to-point) support:*** Otro protocolo de conexión IP (más versátil) en líneas serie. A continuación, aparecerán preguntas adicionales sobre tarjetas de red. Si dispone de alguna, búsquela en la figura 1 y elija la opción correspondiente.

- ***Sony CDU31A/CDU33A CDROM driver support:*** A partir de esta opción aparecerán una serie de preguntas para seleccionar su tipo de cdrom, si dispone de él. Sólo comentaremos aquí que los poseedores de Mitsumi FX300 ó FX400 no deben elegir un manejador en este apartado, sino la opción IDE/ATAPI al principio de este apartado. Lo mismo vale para poseedores de otros lectores IDE.

En este punto, se procede a elegir todos los sistemas de ficheros que se quiera instalar. Comentaremos sólo los mas interesantes. En el resto, se responderá no, excepto para usuarios que tengan un interés especial.

- ***Standard (minix) fs support:*** El sistema de ficheros minix es el que se empleó originalmente en Linux, y el que se encuentra en los discos de arranque. Aunque se sigue empleando por este motivo, la etiqueta de estándar empieza a quedar obsoleta.

- ***Second extended fs support:*** El sistema ext2 es el verdadero sistema de ficheros nativo de Linux, hoy día. Muy

El proceso de compilación del núcleo requiere algo de tiempo y depende del procesador del sistema

robusto y eficiente, es imprescindible seleccionarlo.

- ***msdos fs support:*** Si desea leer discos msdos, o la partición dos de su disco duro, responda afirmativamente en esta opción.

- ***umsdos: Unix like fs:*** Este sistema es una extensión al sistema msdos, de forma que podemos leer normalmente ficheros de discos DOS (con nombres de 8 caracteres y extensión de 3). Sin embargo, desde Linux, aparece la capacidad de asociar nombres de mayor número de caracteres a los ficheros. Es el sistema utilizado en la distribución mini-linux, y permite su instalación como una estructura de directorios dentro de un sistema DOS. Contra la ventaja de la sencillez de instalación, encontraremos viejos inconvenientes como la fragmentación de sectores y la falta de robustez ante cortes de alimentación.

- ***/proc filesystem support:*** Este sistema es imprescindible en Linux, y permite habilitar la comunicación con el núcleo mediante la lectura y escritura en cierta zona de ficheros, normalmente a partir del directorio */proc.*

- ***NFS filesystem support:*** Esta opción aparece si se han elegido opciones de red, y permite compartir estructuras de ficheros (no sólo unidades completas, sino también subárboles de directorios) por medio de la red TCP/IP.

- ***ISO9660 cdrom filesystem support:*** Si eligió algún tipo de unidad de cdrom, es bastante probable que desee responder afirmativamente. Prácticamente cualquier cdrom está grabado en este formato, y es el que hay que emplear al montarlos.

- ***PS/2 mouse (alias "auxiliary device"):*** Ésta y otras preguntas similares le permitirán elegir el tipo de ratón (bus) disponible en su sistema. Para ratones serie, no es necesario seleccionar ningún manejador. Esta opción concreta (ps2) selecciona el ratón bus más abundante, el que se conecta justo junto al conector de teclado, y tiene el mismo tamaño que éste.

- ***Sound card support:*** Si tiene una tarjeta de sonido, elegir esta opción desencadenará otra serie de preguntas para averiguar las características concretas de su sistema, como tipo de tarjeta, puertos e interrupciones empleadas, etcétera.

El resto de opciones habilitan auditorías en el núcleo Linux. Sólo son útiles si va a realizar modificaciones al núcleo, o tiene problemas con algún controlador (principalmente SCSI) y desea mayor información. Normalmente, se responderá no a todas las opciones.

COMPILACIÓN DEL NÚCLEO BÁSICO

Una vez escogido el perfil de nuestro sistema, procedemos a realizar la compilación. Tenga en cuenta que este proceso requerirá algo de tiempo, y depende del procesador y la cantidad de memoria disponible en su sistema. En general, la suma de RAM y área de *swap* debería superar los 16 megas para no tener problemas a mitad de ejecución.

También es especialmente crítico que su controlador de disco duro se comporte perfectamente en Linux. Errores fatales producidos por el compilador a mitad de proceso ("signal 11" y errores similares) suelen indicar algún tipo de problema en lecturas y escrituras masivas en disco. Dado que la recompilación del núcleo genera un gran trasiego de información hacia y desde el disco duro, y utiliza intensivamente la memoria RAM y caché por parte del compilador, se produce uno de los mayores esfuerzos por parte de su PC. En general, cualquier controladora IDE sin caché y con bus ISA elimina cualquier posible problema de este tipo. Si no puede compilar el núcleo en su propia máquina, pida a un amigo que lo recompile por usted, o procure la configuración mas sencilla posible. Controladoras caché o bus local pueden ser potenciales fuentes de problemas, aunque no es frecuente que ocurra así.

Los siguientes comandos realizarán la compilación:

```
make dep
```

Esto compila la parte del núcleo que depende de la configuración que se ha escogido en el apartado anterior. Cuando acabe este proceso, teclee:

```
make zImage
```

Al concluir este comando, habrá quedado un ejecutable final en *¡arch/i386/boot/zImagei.* Ahora sólo nos resta instalarlo, con:

```
cd arch/i386/boot
chmod 400 zImage
rdev -R zImage 1
cp zImage /k209
```

Esto marca los permisos adecuados en la imagen binaria, y la copia en el directorio raíz. En nuestro caso la hemos denominado ík209í (suponiendo el núcleo 1.2.9), pero el nombre que elija es trivial.

A partir de ahora supondremos que usted arranca con LILO. En otro caso, deberá sustituir el núcleo antiguo por el nuevo tal y como se especifique en la documentación correspondiente.

En el primer caso, use su editor preferido y edite */etc/lilo.conf.* Por cierto, siempre es aconsejable guardar una copia del fichero original por si acaso.

Vamos a modificar "a mano" el arranque. Conseguiremos ciertas ventajas por ello, la mayor de las cuales consiste en conservar el arranque antiguo, por si algo no funciona como es debido. De cualquier manera, siempre es aconsejable tener un disquete de arranque a mano para remediar cualquier inconveniente que se presente.

Si es la primera vez que edita este fichero, observará unas líneas similares a:

```
# Linux bootable partition config
image = /vmlinuz
  root = /dev/hda3
  label = Linux
  read-only
# Linux bootable partition ends
```

Duplique dichas líneas. En las nuevas que acaba de copiar, cambie en el campo *label* la palabra "Linux", la cual usted especificó cuando instaló LILO, por una palabra que identifique el nuevo núcleo. Por ejemplo, teclee "nuevo".

En la línea que comienza por *image,* sustituya "/vmlinuz" por el nombre del fichero imagen. Si sigue nuestro ejemplo, sería "/k209".

Aunque por el momento no es necesario, es posible que necesite paráme-

tros para el arranque, en caso de dispositivos especiales, como los CD de Sony. Tome nota de que se debería añadir una línea, tras la de "Label", con:

append="opciones"

Las opciones van entre dobles comillas. Por ejemplo, "sbpcd=0x230,SoundBlaster", para detectar el cdrom que se engancha directamente a las tarjetas Sound Blaster.

Hay infinidad de posibilidades en cuanto a estas opciones. Si tiene dispositivos que no se detectan adecuadamente, compruebe la documentación que acompaña a los fuentes (por ejemplo, para el cdrom, en */usr/src/linux/drivers/block*).

Cuando acabe de modificar el fichero, grábelo. Desde la línea de comando, teclee:

lilo

Esto debe analizar, compilar e instalar el fichero que se ha modificado, y produce una línea por cada etiqueta de arranque que se haya definido. Nótese que aparece un asterisco en la primera etiqueta, indicando que será la partición por defecto en el arranque. LILO arrancará por defecto con el primer grupo de líneas del tipo de las que usted ha modificado. Así podrá elegir el orden de arranque a su gusto.

Si aparece algún error, revise la sintaxis de lo tecleado con su editor, y vuelva a teclear "lilo". Si ocurre algún desastre con el fichero, siempre puede restaurar la copia del fichero original, o bien invocar "setup" y reconfigurar LILO.

Cuando todo parezca correcto, rearranque su sistema (*shutdown -r now*).

CONCLUSIONES

Si todo funcionó de forma correcta, en estos momentos se está iniciando el nuevo núcleo. Si usted observa alguna anomalía, como por ejemplo que algún periférico ahora no funciona, puede suceder:

- olvidó incluir el manejador adecuado en la configuración.
- faltan parámetros de arranque para que Linux reconozca el dispositivo
- el periférico no está soportado (aún)

```
Kernel math emulation
Floppy disk support
Normal (MFM/RLL) disk and IDE disk support
        Use old disk-only driver for primary i/f
                Include new IDE driver for secondary i/f
        Use New IDE driver for primary/secondary i/f
                Include support for IDE/ATAPI CDROMs
XT harddisk support

Networking support
Limit memory to low 16M
PCI bios support
        PCI bridge optimization
System V IPC
Kernel support for ELF binaries
Use 486 flag for 486-specific optimizations
Set version information on all symbols for modules

IF (Networking support)
        TCP/IP networking
                IP forwarding/gatewaying
                IP multicasting
                IP firewalling
                IP accounting
                PC/TCP compatibility mode
                Reverse ARP
                Assume subnets are local
                Disable NAGLE algorithm
        The IPX protocol

SCSI support
        SCSI disk support
        SCSI tape support
        SCSI CDROM support
        SCSI generic support
        Probe all LUNs on each SCSI device
        Adaptec AHA152x
        Adaptec AHA1542
        Adaptec AHA1740
        Adaptec AHA274x/284x
        Buslogic SCSI
        EATA-DMA (DPT, NEC & ATT)
        UltraStor 14F/34F
        Future Domain 16xx SCSI
        Generic NCR5380
        IF (PCI bios support)
                NCR 53c7, 8xx
        Always IN2000
        PAS16 SCSI
        QLOGIC SCSI
        Seagate ST02 & Future Domain TMC-8xx
        Trantor T128/128F/228
        UltraStor
        7000 FASST

IF (Networking support)
        Network Device Support
                Dummy Net Driver
                SLIP (serial line) support
                        CSLIP compressed headers
                        16 channels instead of 4
                PPP (point-to-point) support
                PLIP (parallel port) support
                Do you want ALPHA test drivers
                Western Digital/SMC cards
                        WD80*3 support
                        SMC Ultra support
                AMD LANCE and Pcnet (AT1500 & EN2100)
                3COM cards
                        3c501 support
                        3c503 support
                        IF (ALPHA drivers)
                                3c505 support
                                3c507 support
                        3c509/3c579 support
                Other ISA cards
```

```
                    Cabletron E21xx support
                    DEPCA support
                    EtherWorks 3 support
                    IF (ALPHA drivers)
                        Arcnet support
                        AT7100 support
                        EtherExpressPro support
                        EtherExpress support
                        NI5210 support
                        NI6510 support
                        WaveLAN support
                    HP PCLAN+ (2747B & 27252A)
                    HP PCLAN (27245 & 27xxx series)
                    NE2000/NE1000
                    SK_G16 support
                EISA, VLB and PCI add on boards
                    IF (ALPHA drivers)
                        Ansel Comm EISA 3200
                    Apricot Xen-II on board
                    DE425, DE434, DE435
                    Zenith Z-Note
                Pocket and portable adaptors
                    AT-LAN-TEC/Realtek pocket adaptor
                    D-Link DE600 pocket adaptor
                    D-Link DE620 pocket adaptor

Sony CDU31a/CDU33a CDROM driver
Mitsumi (not IDE/ATAPI)
Matsushita/Panasonic CDROM driver (Creative CDROM)
        Matsh/Panas second CDROM
            Matsh/Panas third CDROM
                Matsh/Panas fourth CDROM
Aztech/Orchid/Okano/Wearnes (non IDE)
Sony CDU535

Standard (minix) fs
Extended fs support
Second extended fs support
xiafs filesystem
msdos fs support
        umsdos: Unix like fs
/proc filesystem
IF (TCP/IP networking)
        NFS filesystem support
ISO9660 support
OS/2 HPFS support
System V y Coherent filesystem

Cyclades async mux support
Parallel printer support
Logitech busmouse support
PS/2 ("auxiliary device") support
        C&T 82C710 (TI travelmate) support
Microsoft busmouse support
AXIXL busmouse support
QIC-02 tape support
        want runtime configuration for QIC-02
QIC-117 tape support
        number of ftape buffers

Sound card support

Kernel profiling support
        profile shift count
IF (SCSI support)
        Verbose SCSI error reporting
```

Listado 1. Opciones de "make config".

El primer caso puede resultar frustrante si se produce en abundancia, pero siempre tiene sencilla solución: vuelva a repetir todos los pasos desde "*make config*". El segundo caso es frecuente y el lector SONY CD de doble velocidad (CDU31a) es un ejemplo clásico. Dado que Linux realiza una autodetección leyendo y escribiendo en ciertos puertos de hardware, es posible tener problemas cuando hay un dispositivo (por ejemplo una tarjeta de red NE2000) instalada en un puerto en el que intentamos detectar otro (por ejemplo el CDU31a). En este caso concreto, tener ambos periféricos cuelga el ordenador cuando se activa la tarjeta de red mientras se intenta detectar el lector CD, de forma que los responsables del núcleo decidieron deshabilitar la autodetección de éste último. Los parámetros de arranque, suponiendo que el puerto I/O es el 300h, el nivel IRQ es 5, y la controladora es la pequeña tarjeta que acompaña al lector, se describen en la línea (para LILO):

append="cdu31a=300,5,none"

En el tercer caso, la solución ya no está en sus manos. Si tras infructuosos intentos, y leyendo la documentación que acompaña los ficheros fuente, usted descubre que su hardware no está soportado (el no disponer de una línea explícita para el dispositivo en "make config" es una indicio casi definitivo), deberá esperar a que aparezca una versión de núcleo actualizada. En el caso de la controladora PCI SCSI AHA-2940, por ejemplo, a mediados de junio sólo hay parches experimentales que se aplican sobre el núcleo 1.2.8. Quizá en pocas semanas se incluya un soporte oficial en un nuevo núcleo, pero hasta entonces, los poseedores de dichas tarjetas no podrán utilizar los dispositivos que dependan de este controlador.

Esperamos que la configuración de Linux le resulte algo más sencilla tras la lectura de este artículo. Como siempre, quizá deba repetir unas cuantas veces algún paso para dominar completamente la reconstrucción del núcleo. En este caso, la recomendación es: tener paciencia y leer los ficheros README que están distribuidos a lo largo del árbol del núcleo, los cuales podrán arrojar algo de luz si se atasca en algún punto concreto.

Esperamos que sienta la satisfacción que produce descubrir que puede controlar completamente las características de su sistema operativo, donde que las funciones "indocumentadas" están al alcance de su editor de textos. Si es así, Linux será su sistema preferido por mucho tiempo. ■

CAPTURA DE LA SALIDA DE UN COMANDO TSO A FICHERO (2ª PARTE)

José María Peco

El presente artículo pretende ser la continuación de aquel, y tiene por objetivo tratar el fichero obtenido en el mismo, para así poder navegar por el sistema mediante paneles de selección. Para llevar a cabo este objetivo, se va a hacer uso del fichero obtenido al ejecutar el comando LISTA, pero con la opción STATUS, lo cual permite obtener no sólo las librerias asignadas al entorno del usuario sino también el nombre lógico que tienen asociado, y su disposición.

No se comenta el procedimiento para la obtención del fichero secuencial, por ser una modificación del procedimiento comentado el mes pasado. Aquel generaba una línea en salida por cada librería asignada, y éste genera dos como puede verse en la pantalla correspondiente, pero quitando este pequeño detalle, el resto es idéntico.

Procedimiento para pasar a fichero secuencial la salida del comando:

Al llegar a este punto, se dispone de un fichero secuencial con las librerías asignadas al entorno, y, escrito en el mismo registro, el nombre lógico que tienen asignado, y la disposición.

Esta primera parte, lo que hace es ini-

```
                    TSO COMMAND PROCESSOR

ENTER TSO COMMAND, CLIST, OR REXX EXEC BELOW:
 ===> lista status
—DDNAME—DISP—
DESJMP.ISPF.ISPPROF
 ISPPROF  KEEP
TERMFILE  SYSPRINT
TERMFILE  SYSOUT

DESJMP.SPFLOG1.LIST
 ISP14421 CATLG
DESJMP.JMP.PANEL
 ISP14422 KEEP,KEEP
DESJMP.JMP.ESKELETO
 ISP14423 KEEP,KEEP
DESJMP.JMP.MENSAJES
 ISP14424 KEEP,KEEP
DESJMP.INT.TABLA
 ISP14425 KEEP,KEEP
DESJMP.JMP.TMP
 SALIDA   KEEP
*DESJMP.JMP.JCL
 SYS00061 KEEP
***
```

Fig. 1: Lista proporcionada por el sistema al ejecutar el comando : LISTA STATUS.

En el artículo del mes pasado se trataba de capturar la salida de un comando no direccionable a un fichero, cosa que se realizó haciendo uso del comando de TSO que lista las librerías que se encuentran asignadas al entorno del usuario: LISTALL o abreviadamente LISTA.

```
                          Lista de librerías                    ROW 1 TO 16 OF 94
OPCION====>                                                      SCROLL ==> CSR
Opciones : B Browse          <<< LIBRERÍAS ASIGNADAS >>>
           E Edit
           M Miembros
  m                Librerias                    Observaciones
    DESJMP.ISPF.ISPPROF                          ISPPROF
    DESJMP.JMP.ESKELETO                          ISP14423
    DESJMP.JMP.MENSAJES                          ISP14424
    DESJMP.JMP.PANEL                             ISP14422
    DESJMP.JMP.TMP                               SALIDA
    DESJMP.SPFLOG1.LIST                          ISP14421 CATLG
    TERMFILE                                     SYSPRINT
****************************************** BOTTOM OF DATA ********************************************
```

Fig. 2: Pantalla que permite visualizar o editar las librerías listadas en la figura anterior.

cializar las variables EOF de (End-Of-File) para saber cuando se detecta el fin de fichero.

RETCOD de (Return Code) para memorizar el código de retorno devuelto por los servicios de ISPF que se invoquen.

10500 memoriza el código de retorno que ha generado la cesión del control a esta rutina. Dicho código se encuentra contenido en la variable del sistema &LASTCC.

10600 Examina si el código de retorno es 8, y si es así ejecuta las instrucciones contenidas entre las línea 10700-11600.

10800 Examina el contenido de una variable que se carga en el flujo de ejecución normal, y que sirve para detectar el punto desde el que ha bifurcado a esta rutina, ya que el código de retorno 8 es devuelto por más de una función. Concretamente está examinando si la variable PASO contiene la cadena EXAMEN, valor que se carga en la línea 13700.

11000 En el caso de que contenga dicho valor, quiere decir que no existe una fila con la clave que se está buscando, conclusión correcta, ya que se quiere dar de alta una entrada en una tabla indexada, y para ello necesitamos que no exista. Por esta razón es por lo que se carga la variable EXISTE con el valor NO.

11100 Retorna al punto desde el que se bifurcó a esta rutina.

11300 Escribe el mensaje indicando que se quiere crear una tabla ya existente.

11400 Se interrumpe la ejecución de la clist hasta que el usuario pulse INTRO. Esto se consigue indicando al sistema que lea desde el terminal el valor de la variable SEGUIR, la cual no sirve más que para interrumpir la ejecución de la clist, a la espera de que se pulse INTRO.

11500 Se sale de la clist devolviendo el código de retorno 99

11700 En el caso de que el código que provoco esta bifurcación incon-

```
/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<  ENTORNO0 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>  */ 00000100
/* ============= CAPTURA LAS LIBRERIAS ASIGNADAS  ====================  */ 00000200
PROC 0 TEST                                                                  00000300
IF &TEST = TEST THEN CONTROL SYMLIST CONLIST MSG                             00000400
                ELSE CONTROL NOMSG                                           00000500
/*                   CONTROL NOMSG NOCONLIST NOSYM                           */ 00000600
/*                                                                           */ 00000700
/*                                                                           */ 00000800
        CONTROL MSG                                                          00000900
        SET CABECERA = <<< LIBRERIAS ASIGNADAS >>>                           00001000
        SET RC = 0                                                           00001100
        CONTROL NOLIST NOCONLIST                                             00001200
        SET &SYSOUTTRAP = 999                                                00001300
        LISTA STATUS                                                         00001400
        SET &SYSOUTTRAP = 0                                                  00001500
        SET &NOMBRE = TABLA                                                  00001600
        SET &CONTA  = 0                                                      00001700
        SET &ULTIMO = &SYSOUTLINE                                            00001800
        IF &TEST = TEST THEN WRITE NUMERO DE ELEMENTOS: &ULTIMO              00001900
/*                                                                           */ 00002000
/*================== MEMORIZAR SALIDA ==============================      */ 00002100
/*                                                                           */ 00002200
        IF &TEST = TEST +                                                    00002400
          THEN SET &ETIQUETA = EXIT1                                         00002500
          ELSE SET &ETIQUETA = LISTAR                                        00002600
LOOP1: SET &CONTA  = &CONTA + 1                                              00002300
        IF  &CONTA > &ULTIMO THEN GOTO &ETIQUETA                             00002700
        SET     &NOMBRE&CONTA = &&SYSOUTLINE&CONTA                           00002800
        GOTO LOOP1                                                           00002900
/*                                                                           */ 00003000
EXIT1:  SET &CONTA  = 0                                                      00003100
LOOP2: SET &CONTA  = &CONTA + 1                                              00003200
        IF  &CONTA > &ULTIMO THEN GOTO LISTAR                                00003300
        SET     &SALIDA = &&NOMBRE&CONTA                                     00003400
        WRITE &CONTA  &SALIDA                                                00003500
        GOTO LOOP2                                                           00003600
/*                                                                           */ 00003700
LISTAR: SET &FICHERO= '&SYSUID..JMP.SLPGM'                                   00003800
/*                                                                           */ 00003900
/*      ============= ESCRIBIR SALIDA EN FICHERO ==================      */ 00004000
/*                                                                           */ 00004100
        IF &SYSDSN(&FICHERO) = OK THEN +                                     00004200
             ALLOC FILE(SALIDA) DATASET(&FICHERO) OLD REUSE                  00004300
        ELSE ALLOC FILE(SALIDA) DATASET(&FICHERO) NEW CATALOG  +             00004400
                                UNIT(3380) RECFM(F B) LRECL(133) +           00004500
                                BLKSIZE(23408) TRACKS SPACE(3 1) REUSE       00004600
/*                                                                           */ 00004700
        OPENFILE SALIDA OUTPUT                                               00004800
        SET     &SALIDA = 'LA SALIDA CONTIENE:'                              00004900
        PUTFILE  SALIDA                                                      00005000
/*                                                                           */ 00005100
        SET &RC = 0                                                          00005200
        ERROR +                                                              00005300
              DO                                                             00005400
                 SET &RC = &LASTCC                                           00005500
                 IF &RC = 400 THEN +                                         00005600
                   WRITE FIN NORMAL DE LA CLIST                              00005700
                 ELSE  +                                                     00005800
                     DO                                                      00005900
                       WRITE INESPERADO FIN DE CLIST                         00006000
                       WRITE RC ES &RC                                       00006100
                       WRITE &SALIDA                                         00006200
                       EXIT                                                  00006300
                     END                                                     00006400
              END                                                            00006500
/*                                                                           */ 00006600
        SET     &CONTA  = 1                                                  00006700
        SET     &BLANCOS= &STR(                                             )00006800
        IF &TEST = TEST THEN WRITE ENTRO A GRABAR EN FICHERO                 00006900
GRABAR: SET     &CONTA  = &CONTA + 1                                         00007000
        IF      &CONTA  > &ULTIMO THEN GOTO CERRAR                           00007100
        SET     &LINEA1 = &&NOMBRE&CONTA                                     00007200
        SET     &LINEA1 = &STR(&LINEA1 &BLANCOS)                             00007300
        SET     &LINEA1 = &STR(&SUBSTR(1:44,&STR(&LINEA1)))                  00007400
        SET     &LINEA2 = &STR()                                             00007500
        IF &STR  (&SUBSTR(1:8,&STR(&LINEA1))) = TERMFILE +                   00007600
          THEN  DO                                                           00007700
/*                SET &LINEA1(= &STR(&SUBSTR(01:10,&STR(&LINEA1)))         */ 00007800
                  SET &LINEA2 = &STR(&SUBSTR(09:23,&STR(&LINEA1)))           00007900
                  GOTO SEGUIR                                                00008000
                  END                                                        00008100
        SET     &CONTA  = &CONTA + 1                                         00008200
        IF      &CONTA  > &ULTIMO THEN GOTO CERRAR                           00008300
        SET     &LINEA2 = &&NOMBRE&CONTA                                     00008400
        SET     &LINEA2 = &STR(&LINEA2 &BLANCOS)                             00008500
        SET     &LINEA2 = &STR(&SUBSTR(1:44,&STR(&LINEA2)))                  00008600
```

```
/*       SET ELEMENTO    = &STR(&SUBSTR(01:44,&LINEA1))                    */ 00008700
/*       SET COMENTA     = &STR(&SUBSTR(01:15,&LINEA2))                    */ 00008800
SEGUIR: SET &SALIDA      = &STR(&LINEA1&LINEA2)                              00008900
         PUTFILE  SALIDA                                                     00009000
         IF &TEST = TEST THEN WRITE &SALIDA                                  00009100
        GOTO   GRABAR                                                        00009200
CERRAR: CLOSFILE SALIDA                                                      00009300
        FREE FILE(SALIDA)                                                    00009400
/*                                                                           */ 00009500
/* ============== MUESTRA LA TABLA DE ELEMENTOS ====================       */ 00009600
/*                                                                           */ 00009700
CONTROL MSG                                                                  00009800
SET EOF = NO                                                                 00009900
SET RETCOD = 0                                                               00010100
ERROR +                                                                      00010300
   DO                                                                        00010400
   SET RETCOD = &LASTCC                                                      00010500
   IF &RETCOD = 8   THEN +                                                   00010600
     DO                                                                      00010700
        IF &PASO = EXAMEN THEN +                                             00010800
          DO                                                                 00010900
             SET EXISTE = NO                                                 00011000
             RETURN                                                          00011100
          END                                                                00011200
        WRITENR YA EXISTE LA TABLA ENTORNO0. NO PUEDO CREARLA                00011300
        READ SEGUIR                                                          00011400
        EXIT (RETCOD)                                                        00011500
     END                                                                     00011600
   IF &RETCOD = 400 THEN +                                                   00011700
     DO                                                                      00011800
      SET EOF = SI                                                           00011900
      RETURN                                                                 00012000
     END                                                                     00012100
  ELSE +                                                                     00012200
     EXIT (RETCOD)                                                           00012300
  END                                                                        00012400
ALLOC FI(ENTRADA) DA(&FICHERO) SHR REUSE                                     00012500
ISPEXEC TBCREATE ENTORNO0 NOWRITE NAMES(ELEMENTO,COMENTA)                    00012600
OPENFILE ENTRADA                                                             00012700
/*                                                                 */        00012800
     GETFILE  ENTRADA                                                        00012900
/*   SET LIBRERIA = &STR(&SUBSTR(1:45,&ENTRADA))                   */        00013000
/*                                                                 */        00013100
GETFILE  ENTRADA                                                             00013200
DO WHILE &EOF = NO                                                           00013300
    SET ELEMENTO = &STR(&SUBSTR(01:44,&ENTRADA))                             00013400
    SET COMENTA  = &STR(&SUBSTR(45:60,&ENTRADA))                             00013500
/*                                                                 */        00013600
    SET PASO   = EXAMEN                                                      00013700
    SET EXISTE = SI                                                          00013800
    ISPEXEC TBEXIST ENTORNO0                                                 00013900
    IF &EXISTE = NO THEN ISPEXEC TBADD ENTORNO0 SAVE(ELEMENTO,COMENTA)       00014000
   GETFILE ENTRADA                                                           00014100
END                                                                          00014200
CLOSFILE ENTRADA                                                             00014300
FREE  FI(ENTRADA)                                                            00014400
ERROR OFF                                                                    00014500
ISPEXEC TBSCAN  ENTORNO0 ARGLIST(ELEMENTO) NEXT                              00014600
ISPEXEC TBSORT  ENTORNO0 FIELDS(ELEMENTO,C,A)                                00014700
ISPEXEC LIBDEF ISPPLIB DATASET ID('&SYSUID..JMP.PANEL')                      00014750
SET &SIGUE = SI                                                              00014800
DO WHILE &SIGUE = SI                                                         00014900
    ISPEXEC TBTOP   ENTORNO0                                                 00015000
    ISPEXEC TBDISPL ENTORNO0 PANEL(ENTORNO0)                                 00015100
    IF &LASTCC = 8 THEN -                                                    00015200
       DO                                                                    00015300
         ISPEXEC TBCLOSE ENTORNO0                                            00015400
         EXIT (99)                                                           00015500
      END                                                                    00015600
    SET &FICHERO  = &ELEMENTO                                                00015900
    IF &MARCA  = B THEN -                                                    00016000
      DO                                                                     00016100
          ISPEXEC BROWSE DATASET('&FICHERO')                                 00016200
          SET &COMENTA = 'VISUALIZADA'                                       00016300
      END                                                                    00016400
    IF &MARCA  = E THEN -                                                    00016500
      DO                                                                     00016600
          ISPEXEC EDIT   DATASET('&FICHERO')                                 00016700
          SET &COMENTA = 'EDITADA    '                                       00016800
      END                                                                    00016900
   IF &MARCA  = M THEN -                                                     00017000
     DO                                                                      00017100
        EXEC '&SYSUID..OPER.OPERCLIB(ENTORNOM)','&FICHERO'                   00017200
     END                                                                     00017300
   SET &MARCA = &STR()                                                       00017400
 END                                                                         00017500
```

dicional no fuera el 8 (línea 10600) examina si es el código 400, generado cuando en la lectura de un fichero se ha detectado el fin de fichero.

11900 si se ha detectado esta condición, se carga la variable de usuario EOF con el valor SI y se regresa con RETURN al punto desde el que se bifurcó a esta rutina de errores.

12300 En el caso de no ser ninguno de estos errores, sale de la clist devolviendo el código de retorno que ha provocado el error.

Los dos siguientes grupos de sentencias constituyen el procedimiento para el que se ha escrito la rutina de errores anterior. Sólo queremos comentar que una tabla ISPF no es sino una tabla o fichero indexado de doble entrada; tiene tantas filas como registros, y cada registro tiene tantos campos como filas tenga definida la tabla.

12500 Asigna el dataset cuyo nombre se encuentra contenido en la variable FICHERO para ser usado de entrada con disposición SHARED, y le asigna el nombre lógico de ENTRADA.

12600 Llama al servicio de ISPF : TBCREATE para crear una tabla ISPF temporal (NOWRITE) con los campos de nombre ELEMENTO y COMENTA.

ISPF, por haber invocado una de sus funciones, nos devuelve siempre un código de retorno.

Si este código de retorno no es 0 salta a la rutina de errores que se ha comentado. Como lo normal es que en este caso sea 0, pues no existe la tabla ya que es en memoria, sigue en secuencia.

Si ya existiera la tabla, devuelve el código de retorno 8. de ahí el mensaje de la línea 11300.

12700 Abre el fichero ENTRADA en modo INPUT.

12900 Lee el primer registro, que no interesa ya que solo contiene un texto de cabecera fijo (línea 4900 y 5000).

11000 Lee el siguiente registro, el cual ya contiene información válida.

Por lo tanto, con un registro válido leído, se entra en el bucle siguiente:

13300 Define un estructura repetitiva que se ejecutará mientras el

contenido de EOF sea NO

13400 Carga en la variable ELEMENTO las posiciones de 1 a 44 de la variable ENTRADA (buffer que contiene el registro leído)

13500 Carga en la variable COMENTA las posiciones 45 a 60 de ENTRADA.

13700 Carga en la variable PASO la cadena EXAMEN. Esta variable es la que se examina en la rutina de ERROR (10800).

13800 Carga en la variable EXISTE la cadena SI

13900 Llama a la función de ISPF : TBEXIST para saber si existe una fila (registro) en la tabla ispf de nombre ENTORNO0.

Llegados a este punto se puede producir una situación de excepción, pues puede existir o no una fila con el valor contenido en la variable ELEMENTO, dependiendo de que estén duplicados los nombres de las librerías, con lo que esta función puede retornar:

- el código 0 si existe la fila, y sigue en secuencia, o
- el código 8 si no existe la fila, y por tanto bifurca a la rutina de error. Una vez en esta rutina, pregunta por el código, y como sabe de donde viene por el contenido de la variable PASO, carga el valor NO en la variable EXISTE y retorna a la siguiente sentencia a la que provocó su bifurcación. Es decir, retorna a la sentencia 14000

14000 Pregunta por el contenido de la variable EXISTE.

En el caso de que el contenido sea NO, es decir no existe una fila con la entrada dada, llama a la función de ISPF: TBADD para añadir la fila a la tabla, con los campos ELEMENTO Y COMENTA.

14100 Lee un nuevo registro del fichero ENTRADA

14200 Cierra el bucle de la repetitiva DO-WHILE. Este bucle se repetirá hasta que al leer un nuevo registro se detecte el fin de fichero, con lo que se bifurcará de nuevo a la rutina de error, y al comprobar que el código de retorno es 400, cargar en la variable EOF el valor SI, por lo que al retornar de nuevo a este punto, e ir a la cabecera de la repetitiva, comprobará que el contenido de la variable EOF no es NO, y saltará a la línea 12100.

```
)ATTR default($+_)
  ? TYPE(INPUT)  INTENS(low)  CAPS(off) JUST(LEFT)
  # TYPE(OUTPUT)  INTENS(high) CAPS(ON) JUST(LEFT) pad(_)
)BODY
$------------------------------------ Lista de librerías ------------------------------------
$ OPCION====>_Zcmd                                                    +SCROLL ==> _scin+
$
+ Opciones :$B+Browse                    #cabecera                                  +
+            $E+Edit
+            $M+Miembros
+    m                    Librerias                                  Observaciones
-----------------------------------------------------------------------------------------
)MODEL ROWS(ALL)
+      _z?elemento                                                  +?comenta +
)INIT
  &SCIN= Csr
  .zvars  = marca      &marca  = ' '
)END
```

14300 Cierra el fichero asociado al nombre lógico ENTRADA

14400 Libera para el sistema el fichero físico asociado al nombre lógico ENTRADA.

14500 Desactiva la rutina de error.

Así pues, al llegar a este punto ya se dispone de una tabla ISPF cargada con los datos que interesan.

Las siguientes sentencias presentan la parte de la clist que muestra la tabla en un panel, y permite tratar la información contenida en cada una de sus líneas.

14600 establece como argumento de búsqueda para un SCAN , el campo ELEMENTO, y la búsqueda se realizará hacia delante.(NEXT)

14700 Clasifica la tabla por el campo ELEMENTO, en ascendente, y su contenido, aunque sea numérico, será tratado como caracteres alfanuméricos.

14750 Define como librería de paneles la librería especificada.

El siguiente fragmento contiene una repetitiva a fin de tratar todos los elementos de la tabla.

14800 Inicializa la variable SIGUE con el valor SI

14900 Inicia un bucle que se ejecutará mientras la variable SIGUE contenga el valor SI

15000 posiciona el puntero en la primera fila de la tabla clasificada.

15100 LLama a la función de ISPF para presentar en un panel las filas de la TABLA.

ISPF ya se encarga de la paginación, número de filas por página, etc.

15200 Al no haber una rutina de error a la que bifurcar, después de cada llamada se debe preguntar por el código de retorno que devuelve ISPF. Esto es lo que se hace en esta línea.

En el caso de que el código sea 8 , es decir no hay filas, cierra la tabla (15400), y sale de la clist.

15900 Carga la variable FICHERO con el contenido del campo ELEMENTO de la fila seleccionada.

16200 llama a la función de ISPF : BROWSE pasándole como argumento el contenido de la variable FICHERO.

16300 Mueve el texto VISUALIZADA en la variable COMENTA de la fila seleccionada.

16700 llama a la función de ISPF : EDIT pasándole como argumento el contenido de la variable FICHERO.

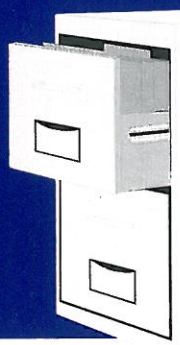
16800 Mueve el texto EDITADA en la variable COMENTA de la fila seleccionada.

17200 Llama a la clist ENTORNOM, pasándole como argumento el contenido de la variable FICHERO con el fin de presentar los miembros de una librería, y tratarles de forma parecida a como se hace con este panel.

17400 mueve blancos a la variable MARCA

17500 Cierra el bucle de la repetitiva.

Por ultimo, se presenta la definición del panel ENTORNO0, que para poder ser tratado, debe ser un miembro de la librería de paneles del usuario: &SYSUID..JMP.PANEL (línea 14750), donde &SYSUID es una variable del sistema que contiene el identificador del usuario de TSO. ■



CONCEPTOS BÁSICOS 1

Juan Manuel y Luis Martín

El presente artículo pretende ofrecer una introducción a los conceptos básicos de Clipper, con el fin de que aquellos lectores cuyos conocimientos sobre el lenguaje sean muy escasos o nulos, puedan iniciarse con uno de los lenguajes de programación más populares del mercado.

Clipper es el lenguaje de programación orientado al manejo y gestión de bases de datos más popular del mercado. Desde su nacimiento como compilador de programas realizados en dBASE, hasta la versión 5.2, ha experimentado una gran evolución, convirtiéndose en un lenguaje de programación independiente. A lo largo de esta evolución se han ido incorporando numerosas y potentes herramientas para facilitar la labor del programador, y dotar a las aplicaciones de un interfaz con el usuario mucho más cómodo y agradable. Este artículo pretende dar a conocer los conceptos básicos de programación con este lenguaje para aquellos lectores que deseen iniciarse en el mismo. En posteriores artículos se realizará una descripción relacionada con los conceptos básicos sobre el tratamiento y manipulación de bases de datos.

CREACIÓN DE PROGRAMAS

Las aplicaciones creadas con Clipper se componen de uno o más módulos fuente que contienen subprogramas o subrutinas. Estos ficheros, una vez han sido compilados, se enlazan entre sí, y con las librerías de Clipper para formar un fichero ejecutable.

La edición de los programas fuente debe realizarse sobre ficheros de texto con la extensión .PRG. Para ello, es posible utilizar cualquier procesador de texto del mercado, incluido el editor PE, que se suministra con el paquete.

Clipper permite insertar comentarios dentro de los archivos fuente, con el fin de clarificar las operaciones que se van realizando. Para ello pueden utilizarse comentarios hasta fin de línea (//) o de tipo bloque (/* */).

```
// Esto es un comentario
/* Y esto
también  */
```

La compilación de un archivo fuente se realiza mediante el comando:

clipper *FicheroFuente [/Opciones]*

Este comando asume por defecto la extensión .PRG del fichero fuente, y genera el código objeto en un fichero con el mismo nombre y la extensión .OBJ. Cada una de las opciones de compilación debe ir precedida por el carácter *slash* (/).

Para generar el fichero ejecutable (.EXE), es necesario enlazar el código objeto con las librerías. Para ello se utiliza el siguiente comando:

rtlink fi *FicherosObjeto* ***li*** *Librerías*

La lista de ficheros objeto (.OBJ) que componen el programa se especifica en la cláusula *FILE* (fi), separando éstos por comas. La lista de librerías (.LIB) que utiliza el programa se especifica en la cláusula *LIBRARY* (li). Sólo será necesario indicar las librerías externas, ya que Clipper incluye directamente las librerías estándar *CLIPPER.LIB*, *EXTEND.LIB* y *DBFNTX.LIB*.

TIPOS DE DATOS

Los tipos de datos que maneja Clipper se muestran en la tabla 1. En general, estos tipos coinciden con los de la mayoría de lenguajes y programas orientados a la gestión de bases de datos. Los operadores aplicables a estos tipos de datos pueden ser consultados en la tabla 2.

Las cadenas pueden contener cualquier carácter del código *ASCII* extendido, y deben ir encerradas entre comillas simples o dobles. El tratamiento de los datos de tipo numérico es muy similar al de otros lenguajes, si bien Clipper no hace distinción entre enteros, decimales, etc.

```
? 10 + 3                    // Visualiza 13
? "Lobo" + "Bueno" // Visualiza: LoboBueno
```

Los valores "verdadero" y "falso" del tipo lógico se representan con *.T.* y *.F.*, respectivamente. Las expresiones de comparación entre operandos de cualquier tipo producen un resultado de tipo lógico. Estas expresiones pueden enlazarse entre sí utilizando los operadores lógicos *.AND.*, *.OR.* y *.NOT.*.

```
? 10 > 1 .AND. 5 < 3 // Visualiza: .F.
? "Lob" $ "LoboBueno" // Visualiza: .T.
```

Los datos de tipo fecha se generan a partir de cadenas de caracteres, utilizando para ello la función de conversión *CTOD()*.

El formato de las fechas y sus separadores se especifican con el mandato *SET DATE FORMAT*.

```
SET DATE FORMAT "dd/mm/yy"
? CTOD( "31/12/95") + 1
// Visualiza: 01/01/96
```

El tipo de datos *Memo* permite almacenar textos de gran tamaño, si bien su existencia se encuentra supeditada a campos de bases de datos.

Por último, Clipper proporciona un tipo de datos nulo (*NIL*) que permite trabajar con variables no inicializadas.

Clipper proporciona multitud de mandatos y funciones para el tratamiento de cada uno de los tipos de datos, que pueden ser consultadas en la ayuda proporcionada por las *Guías Norton*.

ESTRUCTURAS DE CONTROL

Clipper permite utilizar cuatro estructuras de control diferentes para alterar el flujo de los programas. En general, estas estructuras también coinciden con las del resto de lenguajes estructurados. El cuadro 1 muestra la sintaxis de cada una de ellas.

Clipper también incluye una función basada en la estructura IF, que permite trabajar con expresiones en vez de con sentencias:

```
IF( Condición, ExpVerdadera, ExpFalsa )
```

Se trata de una función que devuelve distintos valores dependiendo del cumplimiento de una condición lógica. Los valores devueltos pueden ser de cualquier tipo de datos.

VARIABLES

Con Clipper no es necesario declarar al principio del programa las variables que se van a utilizar. Sin embargo, es una práctica recomendable hacerlo. La opción /W del compilador genera mensajes de aviso (*warnings*), cuando se detectan variables no declaradas. La sintaxis para la declaración de variables es la siguiente:

```
SENTENCIA Identificador; [[:=Inicializador],...]
```

Las posibles sentencias a utilizar son *PUBLIC*, *PRIVATE*, *LOCAL*, *STATIC*, *FIELD* y *MEMVAR*. Como puede verse, es posible inicializar las variables en el momento de la declaración, usando para ello una expresión de cualquier tipo de datos. Si no se inicializa explícitamente, la variable es automáticamente inicializada al valor *NIL*.

Las variables públicas y privadas, sólo se conservan por compatibilidad con versiones anteriores del lenguaje, por lo que su uso no es muy recomendable.

Las variables locales y estáticas son visibles solamente desde el subprograma donde se han creado, ya que para ser visibles desde cualquier módulo deben ser declaradas fuera de cualquier subrutina. Sin embargo, mientras que las variables locales se destruyen cuando ésta ha finalizado, las variables estáticas existen durante toda la ejecución del programa. Además, el valor almacenado en estas últimas no es inicializado en las sucesivas invocaciones del subprograma.

Como Clipper es un lenguaje orientado al manejo de bases de datos, es

CUADRO 1

```
Bucle de tipo "MIENTRAS"
[DO] WHILE Condición
  [Sentencias]
  [EXIT]
  [sentencias]
  [LOOP]
  [Sentencias]
END[DO]

Bucle de tipo "PARA"
FOR Contador = ValorInicial TO valorFinal [STEP
Incremento]
  [Sentencias]
  [EXIT]
  [Sentencias]
  [LOOP]
  [Sentencias]
NEXT

Bifurcación Condicional Simple
IF Condición
  [Sentencias]
[ELSEIF Condición]
  [Sentencias]
...
[ELSE]
  [Sentencias]
END[IF]

Bifurcación Condicional Múltiple
DO CASE
  CASE Condición
    [Sentencias]
  ...
  [OTHERWISE]
    [Sentencias]
END[CASE]
```

Clipper permite utilizar las cuatro estructuras de control comunes a la mayoría de lenguajes

TABLA 1: (Tipos de datos)

Tipo de dato	Descripción
Caracteres	Cadenas de caracteres ASCII (de 0 a 65535)
Numérico	Números reales (entre 10^-308 y 10^308)
Lógico	Valores booleanos (.T. y .F.)
Fecha	Días del calendario (de 1/1/100 a 31/12/2999)
Memo	Textos de tamaño variable (solo campos)
NIL	Valor nulo para variables no inicializadas

posible utilizar las sentencias FIELD y MEMVAR para indicar al compilador si un identificador se refiere a un campo de una base de datos, o a una variable de memoria. Estas sentencias se utilizan en caso de conflicto.

FUNCIONES Y PROCEDIMIENTOS

Clipper es un lenguaje estructurado, por lo que los programas estarán constituidos por subrutinas. Es posible utilizar dos tipos de subrutinas, funciones y procedimientos. La diferencia estriba en que las primeras permiten devolver un valor de cualquier tipo de datos (incluido *NIL*), por lo que pueden ser integradas dentro de expresiones. La sintaxis de ambos tipos de subrutinas es la siguiente:

```
[STATIC] FUNCTION Nombre( [ListaParámetros] )
  [DeclaraciónVariables]
  ...
  [SentenciasEjecutables]
  ...
RETURN [ExpresiónRetorno]

[STATIC] PROCEDURE Nombre( [ListaParámetros] )
  [DeclaraciónVariables]
  ...
  [SentenciasEjecutables]
  ...
[RETURN]
```

Las subrutinas declaradas como estáticas sólo pueden ser llamadas desde el módulo o fichero donde se encuentran implementadas.

El paso de argumentos se realiza *por valor*, es decir, pasando a la subrutina una copia de los valores de los argumentos especificados en la llamada. Sin embargo, es posible pasar los argumentos *por referencia*, anteponiéndoles en la llamada el operador @. En este último caso, las alteraciones realizadas en los parámetros dentro de la subrutina se verán reflejadas en los valores originales de sus respectivos argumentos. El paso de argumentos por referencia puede utilizarse para que una función pueda devolver más de un valor.

El arranque de un programa en Clipper se realiza desde la primera subrutina, llamada programa principal. No es necesario declarar el programa principal, ya que Clipper engloba las sentencias que se encuentran fuera de toda subrutina y las asigna a un procedimiento con el mismo nombre que el módulo. La opción /N del compilador anula esta asignación, y debe usarse cuando se ha definido explícitamente una subrutina principal.

TABLA 2: Operadores

Operador	Descripción
+	Suma o concatenación de cadenas
-	Resta o concatenación especial de cadenas
*	Multiplicación
/	División
^ o **	Exponenciación
%	Resto o módulo
++	Incremento
—	Decremento
=	Comparación de igualdad
==	Comparación de igualdad exacta
!=,<>,#	Comparación de desigualdad
<	Comparación de menor
<=	Comparación de menor o igual
>	Comparación de mayor
>=	Comparación de mayor o igual
$	Inclusión de una cadena en otra
.NOT.,!	Negación
.AND.	Conjunción ("Y" lógico)
.OR.	Disyunción ("O" lógico)
:=	Asignación en lónea
+=	Suma y asignación en línea
-=	Resta y asignación en línea
*=	Multiplicación y asignación en línea
/=	División y asignación en línea
^=	Exponenciación y asignación en línea
%=	Módulo y asignación en línea

MATRICES

Clipper dispone del tipo de datos matriz, aunque su utilización difiere de otros lenguajes. Se trata de matrices dinámicas, ya que su tamaño y dimensiones pueden ser alterados en tiempo de ejecución. Además, los tipos de datos de sus elementos no tienen por que ser uniformes, siendo posible mezclar dentro de la misma matriz números, fechas, cadenas, etc.

La declaración de matrices se realiza de la misma manera que el resto de las variables. Sin embargo, es posible indicar el número de elementos inicial de cada dimensión utilizando la siguiente sintaxis:

```
SENTENCIA NombreMatriz[NElem1,NElem2,...]
SENTENCIA NombreMatriz[NElem1][NElem2]...
```

Además, es posible inicializar las matrices haciendo uso de *matrices literales*. Se trata de matrices cuyos elementos se describen explícitamente, separándolos por comas y encerrándolos entre llaves.

```
LOCAL Matriz := {2,"Perro",.T.,CTOD("20/06/95")}
? Matriz[ 2 ]          // Visualiza: Perro
```

La forma en que Clipper permite la utilización de matrices multidimensionales se basa, realmente, en que cada elemento de una matriz puede, a su vez, ser otra matriz. De esta forma es posible crear matrices cuyos elementos son submatrices, cuyos elementos son a su vez submatrices, y así sucesivamente.

```
Matriz := { 1, { 2, 3 } }
? Matriz[ 1 ]          // Visualiza: 1
Matriz[ 1 ] := { 4, 5 }
? Matriz[ 1, 2 ]       //Visualiza: 5
```

TABLA 3: Mandatos para el redireccionamiento de la salida.

Mandato	Descripción
SET CONSOLE ON|OFF	Activa/Desactiva la salida a pantalla
SET PRINTER ON|OFF	Activa/Desactiva la salida a impresora
SET DEVICE TO SREEN|PRINTER	Redirecciona los mandatos SAY a pantalla o impresora
SET PRITER TO Dispositivo	Redirecciona la salida de la impresora a otro dispositivo
SET ALTERNATE TO Fichero	Redirecciona la salida a un fichero de texto
SET ALTERNATE ON|OFF	Activa/Desactiva la salida a fichero

ENTRADA/SALIDA

Hasta la versión actual, Clipper sólo permite trabajar con pantallas en modo texto. Las operaciones de salida a pantalla pueden agruparse en dos modos:

- Modo *consola*: Los mandatos y funciones escriben a partir de la posición actual del cursor, desplazando la pantalla hacia arriba según se van completando la líneas.

- Modo *pantalla completa*: Los mandatos y funciones van acompañados de dos valores que indican la fila y la columna de la pantalla donde debe comenzar la escritura.

De entre los mandatos y funciones del modo consola, los más utilizados son *??* y *?*. El primero permite visualizar valores de cualquier tipo de datos en la posición actual del cursor. La lista de valores debe ir separada por comas. El mandato ? realiza la misma operación, si bien previamente se realiza un salto de línea.

```
? "La fecha de hoy es", DATE()
```

El mandato más utilizado del modo pantalla completa es *@...SAY*, que visualiza un valor de cualquier tipo en la posición indicada de la pantalla. La sintaxis es la siguiente:

```
@ Fila, Columna SAY Expresión
  [PICTURE Mascara] [COLOR Color]
```

Las cláusulas PICTURE Y COLOR permite dotar a la salida de un determinado formato y color. Sus posibles valores pueden consultarse en las Guías Norton.

Clipper dispone de funciones que permiten controlar el color, editar variables, dibujar recuadros, e incluso crear menús deslizantes de forma rápida y sencilla.

Las matrices en Clipper son dinámicas y pueden variar de tamaño durante la ejecución

La forma más común de introducir datos por teclado es la utilización del mandato *@...GET*. Se trata de un mandato que permite editar en pantalla el contenido de una variable. Para ello, se presenta una plantilla sobre la que puede editarse el valor de la variable. Realmente, este mandato se limita a definir la plantilla de edición. Su activación se realiza mediante el mandato *READ*. De esta forma, es posible editar varias variables a la vez con un solo mandato de este tipo.

```
Texto := "BUENOS DÍAS"
Numero := 100
@ 0, 0 GET Texto
@ 1, 0 GET Numero
READ
```

Igualmente, Clipper dispone de mandatos y funciones que permiten controlar las pulsaciones del teclado, permitiendo así la entrada de datos manualmente. Además, es posible programar determinadas teclas para que realicen tareas específicas.

La impresión de datos con Clipper se basa principalmente en el *redireccionamiento* de los mandatos de pantalla hacia otro dispositivo (archivo o impresora). La tabla 3 muestra los mandatos relacionados con el redireccionamiento de la salida.

La impresión se realiza redireccionando mandatos de pantalla

```
? "Este mensaje aparece sólo en la pantalla"
SET PRINTER ON
? "Este mensaje aparece en pantalla e impresora"
SET CONSOLE OFF
? "Este mensaje aparece sólo en la impresora"
SET CONSOLE ON
SET PRINTER OFF
```

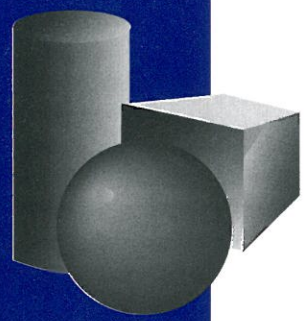
CONCLUSIÓN

Partiendo de los conceptos básicos descritos en este art'culo, es muy recomendable que aquellos lectores que deseen iniciarse con Clipper practiquen desarrollando pequeños programas. En posteriores artículos se realizará una descripción del tratamiento de bases de datos realizado por Clipper. ■

BIBLIOGRAFÍA

"CA-Clipper 5.2"; L. y J.M. Martín Caballero; Ed. R.P.H. 1995.

"CA-Clipper: Guía del Programador"; Computer Associates



SISTEMAS DE VENTANAS EN C++

Ignacio Cea

La potencia de los sistemas de ventanas se basa en la posibilidad de tener simultáneamente a la vista cosas que, de otra forma requerirían el uso de varias pantallas. En este artículo se completarán de una forma muy práctica las técnicas de dibujado en los sistemas de ventanas.

La construcción de cualquier sistema de ventanas pasa por la resolución de tres grandes problemas: La reconstrucción del mismo ante cambios en las ventanas que lo integran, la posibilidad de dibujar en cualquiera de los rectángulos visibles, y la gestión de los eventos originados en los periféricos del ordenador (movimientos y *clicks* en el ratón, pulsación de teclado, etc...).

La anterior entrega ya se ocupó del primero de los problemas, analizando y comparando técnicas y algoritmos que hacían posible la reconstrucción completa de un sistema de ventanas ante variaciones en las características de cualquiera de ellas (creación, destrucción, movimiento y cambio en el tamaño).

Este artículo tratará con el segundo de los problemas; esto es, las técnicas que permiten dibujar en cualquiera de las ventanas del sistema sea cual sea su tamaño y localización dentro de éste.

ESCRIBIR EN LAS VENTANAS

Una situación muy habitual durante la ejecución de una aplicación es cuando una ventana está tapando a otra.

La evolución de cualquier aplicación en la que se presente esta situación hará, normalmente, necesario dibujar no sólo en la ventana superior sino también en la que está parcialmente oculta.

Para acceder a la superior de todas, basta limitar la zona de pantalla a un rectángulo y emplear las órdenes de dibujo de su librería gráfica para conseguir la figura deseada; cosa que, por otro lado, es fácilmente realizable en cualquier lenguaje empleando una función de librería gráfica semejante a la "setviewport" del BorlandC++.

Sin embargo, hacerlo en cualquier otra presenta, normalmente, algunos problemas ya que la zona visible no es, en la mayor parte de los casos, un simple y mero rectángulo. Por otro lado, no debe olvidar que esas zonas visibles de formas extrañas pueden ser descompuestas en un número discreto de rectángulos, como ya se vio en la entrega del mes pasado.

UN PEQUEÑO REPASO

En la anterior entrega se propugnó que el sistema de ventanas C++ que se estaba tratando de modelizar debía estar integrado, al menos, por cuatro clases fundamentales: "Ventana", "ZonasEscritura", "NodoZona" y "NodoVentana".

La segunda clase se empleaba para contener una lista de aquellos rectángulos discretos en los que se podían dividir las partes de todas las ventanas tapadas por otra y así poderlas dibujar cuando quedaran nuevamente visibles. El conocimiento de estas zonas es algo que compete única y exclusivamente a la ventana que tapa, razón por la cual existía un objeto del tipo "ZonasEscritura" dentro de la clase "Ventana" al que se llamaba "ZonasTapadas".

Cuando, por ejemplo, se movía la ventana no había más que calcular, qué partes de las zonas ocultas almacenadas por ésta quedaban nuevamente a la vista, y ordenar a las ventanas dueñas que la dibujaran. Para ello se calculaba la parte vista de la intersección de todas ellas con un rectángulo que englobara la nueva posición de la ventana. El método empleado para este cálculo era "zonas_vistas_por" de la clase ZonasEscritura (figura 1).

FIGURA 1: La clase ZonasEscritura

```
class ZonasEscritura {
        private :
                NodoZona *Primero;
                NodoZona *Ultimo;

        public :
                ZonasEscritura (void) {
                Primero = Ultimo = NULL; }
                ~ZonasEscritura ();
                void limpiar_objeto (void);
                void a–adir_zona (int x1,int
                y1,int x2,int y2,
                        Ventana *Prp);
                Ventana *eliminar_zona
                (int &x1,int &y1,int &x2,
                        int &y2);
                inline NodoZona *primero
                (void) {
                        return
                        (Primero); }
                inline NodoZona *ultimo
                (void) {
                        return (Ultimo); }
                void zonas_vistas_por (int
                x1,int y1,int x2,int y2);
                void zonas_tapadas_por
                (int x1,int y1,
                        int x2,int y2);
};
```

FIGURA 2: La clase Ventana

```
class Ventana {
        private :
                static NodoVentana
                *Inferior;
                static NodoVentana
                *Superior;
                static void zonas_ocultas
                (void);
                static void zonas_vistas
                (void);
                void dibujar _antiguas
                zonas_ocultas (void);

                ZonasEscritura
                ZonasTapadas;
                ZonasEscritura
                ZonasVistas;
                int X1,Y1,X2,Y2,

        public :
                Ventana (int x1,int y1,int
                x2,int y2);
                ~Ventana ();
                void mover (int ax,int ay);
                void reescalar (int ax,int ay);
                virtual void dibujar (...) = 0;
                ...
};
```

FIGURA 3: Algoritmo para el dibujo dentro de una ventana

```
void Ventana::dibujar (...)
{
        // Recorre todas las zonas...
    NodoZona *actual = ZonasVistas.Primero ();
        while (actual != NULL) {
                int     x1 = actual -> x1
                        (), y1 = actual ->
                        y1 (),
                        x2 = actual -> x2
                        (), y2 = actual ->
                        y2 ();
                setviewport (x1,y1,x2,y2);
                .... // Aquí se dibuja lo que
                se quiera
                .... // Teniendo en cuenta las
                coordenadas relativas
                actual = actual -> siguiente
                ();
        }
}
```

LA SOLUCIÓN DEL PROBLEMA

La solución del problema de dibujar es, por tanto, muy sencilla. Basta con que dentro de cada objeto de la clase "Ventana" se almacene otro del tipo "ZonasEscritura", que contenga todos aquellos rectángulos discretos en los que puede ser dividido. Cada vez que se pretenda dibujar algo sobre ella bastará recorrer todas esas zonas, limitando la zona de pantalla al mismo y repitiendo la acción de dibujar.

Dentro de la figura 2 se muestra el aspecto de la clase "Ventana" tras esta modificación. La figura 3, por otro lado, muestra el pseudocódigo para realizar una operación de dibujo sobre una ventana.

CALCULANDO LAS ZONAS VISTAS

Cada vez que suceda algo dentro del sistema de ventanas (crear una nueva, destruir otra, moverla o cambiarla de tamaño) las zonas vistas de las ventanas afectadas deben ser reconstruidas si se quiere mantener la integridad de las mismas.

Sin embargo, si en el caso de las zonas tapadas, el recálculo alcanzaba al total de las ventanas, en el de las vistas sólo

FIGURA 4: Algoritmo para el cálculo de zonas vistas

```
void Ventana::zonas_vistas (void)
{
        // Bucle a todas las ventanas salvo ésta
        NodoVentana *nactual =
        Ventana::Inferior;
        while (nactual1 != NULL) {

                // Limpia las zonas actualmente consideradas
                Ventana *vactual1 = nactual -> propietario ();
                vactual1 -> ZonasVistas.limpiar_objeto ();

                // Añade como zona inicial la totalidad
                vactual1 -> ZonasVistas.añadir_zona (vactual1 -> X1,
                        vactual1 -> Y1,vactual1 -> X2,vactual1 -> Y2,
                        vactual1);

                // Calculo que partes no tapan las superiores
                NodoVentana *npartir = nactual1 -> siguiente ();
                while (npartir != NULL) {
                        Ventana *vactual2 = npartir -> propietario ();
                        vactual2 -> ZonasVistas.zonas_vistas_por
                                                (vactual2 -> x1 (),
                                                 vactual2 -> y1 (),
                                                 vactual2 -> x2 (),
                                                 vactual2 -> y2 ());
                        npartir = npartir -> siguiente ();
                }

                // Siguiente ventana
                nactual = nactual -> siguiente ();

        }
}
```

FIGURA 6: Método erróneo para dibujar

```
void Ventana::dibujar (int forma)
{
        // Bucle para recorrer las zonas
        NodoZona *zactual = ZonasVistas.primero ();
        while (zactual != NULL) {

                // Limita la zona de dibujo...
                int     xp1 = zactual -> x1 (), yp1 = zactual -> y1 (),
                        xp2 = zactual -> x2 (), yp2 = zactual -> y2 ();
                setviewport (xp1,yp1,xp2,yp2);

                // Dibuja...
                switch (forma) {
                        case    CIRCULO :
                                ...
                        case RECTA      :
                                ...
                };

                // Siguiente rectángulo...
                zactual = zactual -> siguiente ();

        }
}
```

FIGURA 7: Método inapropiado para dibujar

```
class Ventana {
                ...
                protected :
                ...
                        virtual void dibujar_figura (int f) = 0;
                ...
};

void Ventana::dibujar (int forma)
{
        // Bucle para recorrer las zonas
        NodoZona *zactual = ZonasVistas.primero ();
        while (zactual != NULL) {

                // Limita la zona de dibujo...
                int     xp1 = zactual -> x1 (), yp1 = zactual -> y1 (),
                        xp2 = zactual -> x2 (), yp2 = zactual -> y2 ();
                setviewport (xp1,yp1,xp2,yp2);

                // Dibuja ...
                dibujar_figura (forma);

                // Siguiente rectángulo...
                zactual = zactual -> siguiente ();

        }
}
```

FIGURA 8: Método recomendado para dibujar

```
class Ventana;
class Figura {
    public :
        virtual void dibujar (Ventana *v = NULL) = 0;
};

#include "Figura.hh"
class Circulo : public Figura {
    private :
        int x,y,r;
    public :
        Circulo (int,int,int);
        virtual void dibujar (Ventana *v = NULL);
};

class Figura;
class Ventana {
        ...
        protected :
        ...
          void dibujar (const Figura &);
        ...
};

#include "Figura.hh"
...
void Ventana::dibujar (const Figura &forma)
{
    // Bucle para recorrer las zonas
    NodoZona *zactual = ZonasVistas.primero ();
    while (zactual != NULL) {

        // Limita la zona de dibujo...
        int     xp1 = zactual -> x1 (),
                yp1 = zactual -> y1 (),
                xp2 = zactual -> x2 (),
                yp2 = zactual -> y2 ();
        setviewport (xp1,yp1,xp2,yp2);

        // Dibuja ...
        forma.dibujar (this);

        // Siguiente rectángulo...
        zactual = zactual -> siguiente ();

        }
}
```

que reconstruir una parte de si, tras algún suceso en cualquiera de las otras ventanas del sistema. Esto se apunta, en forma de comentario dentro de los métodos "dibujar_antiguas_zonas_ocultas", y "destructor" de la clase Ventana. En otros sistemas de ventanas esta acción correspondería al evento de repintado mandado sobre las ventanas.

Con la idea basta para entender esta entrega, y ya en la siguiente se asentarán todas las ideas con un ejemplo.

PRÓXIMO CAPÍTULO

Todas las bases para el manejo de la apariencia de un sistema de ventanas han sido ya descritas. En el próximo capítulo se le presentará el código de todas las clases anteriormente mencionadas, y un pequeño ejemplo donde observe la potencia de lo mostrado.

A partir de ese momento se introducirá en el mundo de la gestión de los eventos. Sin duda, la parte más bonita, pero a la vez complicada. ■

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relaccionado con la programación.

P Estoy empezando a programar en C++ y no acabo de entender el funcionamiento del puntero this ¿Podrían aclarármelo un poco? Gracias de antemano y felicidades por la revista.

Carlos Lafuente García
(Barcelona)

R Se trata de una palabra reservada del lenguaje C++. Es un puntero al propio objeto que ha llamado a un determinado método. Otra definición: "es la dirección inicial del objeto para la que fue llamada la función". O lo que es lo mismo, la dirección de la primera variable en la estructura de clase. Realmente no es necesario utilizar *this* dentro de una función miembro (el único lugar utilizable) pues se pueden utilizar los objetos sólo con el nombre.

Para referirse a un determinado elemento dentro de la clase, se utiliza el operador ->, como en *this->datos* . Pero si se desea una referencia a la estructura completa, debe utilizarse *this, que es el uso más habitual de esta palabra.

Un ejemplo sería:

```
punto punto::operator-=(punto p)
{
x-=p.x;
y-=p.y;
return *this;
}
```

En el que se devuelve un puntero al propio objeto.

Las limitaciones de *this* son:

No se puede asignar valores a *this*.

Sí está permitido asignar un valor a **this*.

Se puede retornar **this* por referencia y por valor.

Para programadores en CA-Clipper

Five Product líder

No lo dudes. Este producto es el que necesitas para hacer tu aplicación en Windows. Extremadamente fácil y potente. Resultados inmediatos. Más de 4.000 usuarios en todo el mundo.

FiveWin
Librería de CA-Clipper para Microsoft Windows
Realiza tus aplicaciones en Microsoft Windows con una presentación y funcionalidad totalmente profesionales. Elegido por la revista Americana Reference(Clipper) como el mejor producto para xBase en Windows (enero 94)
Precio promocional: 14.000 pts.

FiveOS2
Librería de CA-Clipper para IBM-OS2
Ahora puedes realizar aplicaciones de gestión para IBM-OS2 con la misma facilidad que para Windows. El entorno IBM-OS2 es el máximo competidor de Windows y se está haciendo muy popular. Aprovecha esta oportunidad.
Precio promocional: 14.000 pts.

FiveDos
Librería de CA-Clipper para MsDos
Un completo entorno de desarrollo para MsDos con ventanas, ratón, controles CUA y ejecución NoModal. Simula totalmente la funcionalidad de Windows y OS2.
Precio promocional: 14.000 pts.

¡ FiveWin en Vídeo !
Aprende ahora rápida y cómodamente la manera de realizar aplicaciones profesionales de gestión en Windows, tan fácilmente como ver una película de vídeo
Volumen 1.....8.000 pts.

Antonio Linares - Software
Urb. El Rosario, Avda. Rosario 34-A. 29600 Marbella - España
Tfno./fax: 95-2834830 BBS: 95-2213374 / 908 453368 voz

Distribuidor Nacional
Mail Simons S.L. Apdo. 2643. 28080 Madrid. Tfno. 91-5634486
BBS: 91-5637872 FAX: 91-5634451 E-mail: bmartinez @ simons.es

FiveWin

CA-Clipper es una marca registrada de Computer Associates, Windows es una marca registrada de Microsoft Corporation, OS2 es una marca registrada de IBM

DBU para Windows: Gestor de ficheros dBase para programadores. Soporta índices Clipper NTX, dBase III, dBase IV, Foxpro (Comix, Six 2.0) y NSX. PVP: 15.000,0.- Ptas. INSTALADOR DE APLICACIONES BAJO WINDOWS: Compresión de ficheros ZIP, generación automática de diskettes, Multilingüe con fuentes en Five Win, Muestra bitmaps durante la instalación, y con sonido. Demo disponible. PVP: 15.000 ptx+IVA FORMACIÓN, ASESORAMIENTO y CURSOS de programación WINDOWS con Five Win en Madrid: ORTIZ DE ZÚÑIGA. Tfno: (91) 575 20 47.

OTROS DISTRIBUIDORES

ARGENTINA
GO CLIPPER BBS& FIVEWIN
Panamericana 3787
1609 Boulogne
ARGENTINA
+541-737-2767 Voz
+54-1-943-0563 Fax
+54-1-790-1906 BBS

BRASIL
RENET TECNOLOGIA LTDA
Rua Itapicuru, 369
cj. 202-05006-000
Sao Paulo, SP
BRAZIL
55-11-872-0423 Voz
55-11-872-4418 Fax
CIS: 102152,3714

CHILE
PROGRAMMER'S HOUSE
Bustos 2157 dpto 13
Providencia Santiago
CHILE
+56-2-2322948 Voz
+56-2-2322948 Fax

CHILE
COMPUDATA
Tucapel 564 of. 77
CONCEPCIÓN
CHILE
+56-41246035 Voz
+56-41542184 Fax

OTROS
MÉJICO-PERU
COLOMBIA
BOLIVIA
VENEZUELA
CUBA, URUGUAY
PARAGUAY
SE BUSCA DISTRIBUIDOR

PORTUGAL
MULTDIGITAL LDA
Pr. Manuel Guedes, 13 - 4º - Sala 17
+351-2-4647050 Voz
+351-2-4647051 Fax
+351-2-4647052 bbs
joaquím boavida @ tail. pt

La última versión la puedes encontrar en las siguientes revistas, CD-ROM's, BBS's o direcciones de INTERNERT: CD-ROM de PC MEDICA, Sólo Programadores, PC-ACTUAL, KENDER, AMS (Mail Simons s.l.) y las siguientes BBS's: bAuHaUs (+34-5-2213374), KENDER (944763506), CIBERLINEA (902-238624)., CIBERNETIX (986-691525). ftp. eunet. es: pub/InterStand/Simons.

De todas maneras, aunque no sea visible, siempre que un objeto llama a una función de una clase, el valor de *this* siempre va incluido como primer parámetro implícito.

P ***Programando en ensamblador, ¿como puedo convertir datos hexadecimales almacenados en cualquier registro a dígitos ASCII utilizables por cualquier función de salida a pantalla?***

Jesús Villalonga (Ávila)

R El procedimiento es más sencillo de lo que parece, pues cada dígito hexadecimal se representa en un *nibble* (parte de 4 bits dentro de un byte), o lo que es igual, dos dígitos por byte. Si el número a convertir es una *word* (2 bytes), pues obtendremos 4 dígitos en hexadecimal.

```
; Convierte el numero almacenado en
;AX en un número hexadecimal ASCII
cadena db       0,0,0,0

push            ax
mov     ah,al   ; Coge el byte bajo
shr     al,1    ; Lo desplaza 4 veces
; para coger el nibble alto
shr             al,1
shr             al,1
shr             al,1
add             al,90h  ; Suma BCD
daa             ; Ajusta la suma
adc             al,40h
; Le suma el valor ASCII de 'A' - 1
        daa
        mov [cadena+2],al
; Salva el dígito obtenido
        mov             al,ah
        and             al,0Fh
; Ahora aísla el nibble bajo
        add             al,90h
; Repite el proceso
        daa
        adc             al,40h
        daa
        mov [cadena+3],al
; Sigue salvando
        pop             ax
        mov             al,ah
; Coge con el byte alto
        push            ax
; y repite de igual manera...
mov             ah,al
shr             al,1
shr             al,1
shr             al,1
shr             al,1
add             al,90h
daa
adc             al,40h
daa
mov [cadena+0],al
mov             al,ah
and             al,0Fh
add             al,90h
daa
adc             al,40h
daa
mov [cadena+1],al
pop             ax
```

P ***En un programa que estoy realizando, necesito dibujar un rectángulo en la pantalla de tamaño variable, pero al ir cambiando su tamaño, la pantalla se va borrando y modificando. He intentado dibujar la figura mediante XORs pero, lógicamente los colores se modifican. Soluciones, gracias.***

Juan José Durán Alonso (Crevillente/Alicante)

R Siempre que se vaya a plasmar una animación en pantalla e interese salvaguardar el fondo, hay que elegir entre dos alternativas: redibujar cada vez el fondo completo antes de plasmar el *sprite* o figura (lento y parpadeante) o previamente al dibujo de la figura, reponer la porción de fondo que vaya a ser modificado por el nuevo gráfico (mucha menos cantidad de gráficos de fondo a redibujar). Para esto último, es necesario salvar la zona de fondo involucrada antes de dibujar el gráfico superpuesto, para así poder reponerla en el siguiente ciclo. El proceso es el siguiente:

1.- Dibujar el fondo o pantalla base.
...
2.- Salvar la zona de pantalla que se vaya a modificar.
...
3.- Reponer zona de pantalla previamente salvada
4.- Salvar la zona de pantalla que se vaya a modificar.
5.- Dibujar el gráfico o *sprite* superpuesto.
6.- Mostrar el resultado
... Volver al paso 3 hasta el fin de la animación ...

P ***En algunos programas de ejemplo incluidos en el curso de ensamblador que están publicando, la zona del programa en donde se define el espacio para la pila, está declarada como db 512 DUP (0) o similar; pero en algún párrafo especificaron que los elementos de la pila deben ser necesariamente palabras o words, pues sólo se pueden guardar elementos de 16 bits (push ax, pop cx, etc.) y no están permitidas instrucciones como push al o pop ch. ¿No es esto una incoherencia? Supongo que no, pero por favor aclárenmelo.***

José Alberto García García (Talavera de la Reina/Toledo)

R Afortunadamente el lenguaje ensamblador es el más libre de todos, y aunque dispone de innumerables controles de compilación, deja en manos del programador casi todas las decisiones. Realmente, las dos sentencias que vienen a continuación son idénticas:

```
db      0,0
dw      0
```

Lo único que hacen es dejar una zona de memoria de 2 bytes sin utilizar (y a 0 en este caso). El compilador sólo utiliza el tipo de definición para controlar un poco las referencias con punteros, y así evitar algunos errores ocasionados por descuidos. En cualquier caso se puede obligar a un direccionamiento a que esté dirigido a una *word* o a un byte mediante el texto BYTE PTR o WORD PTR. La razón por la que se utiliza una definición en bytes en la zona de pila, es para ver claramente escrito el número de bytes reservados en esa zona.

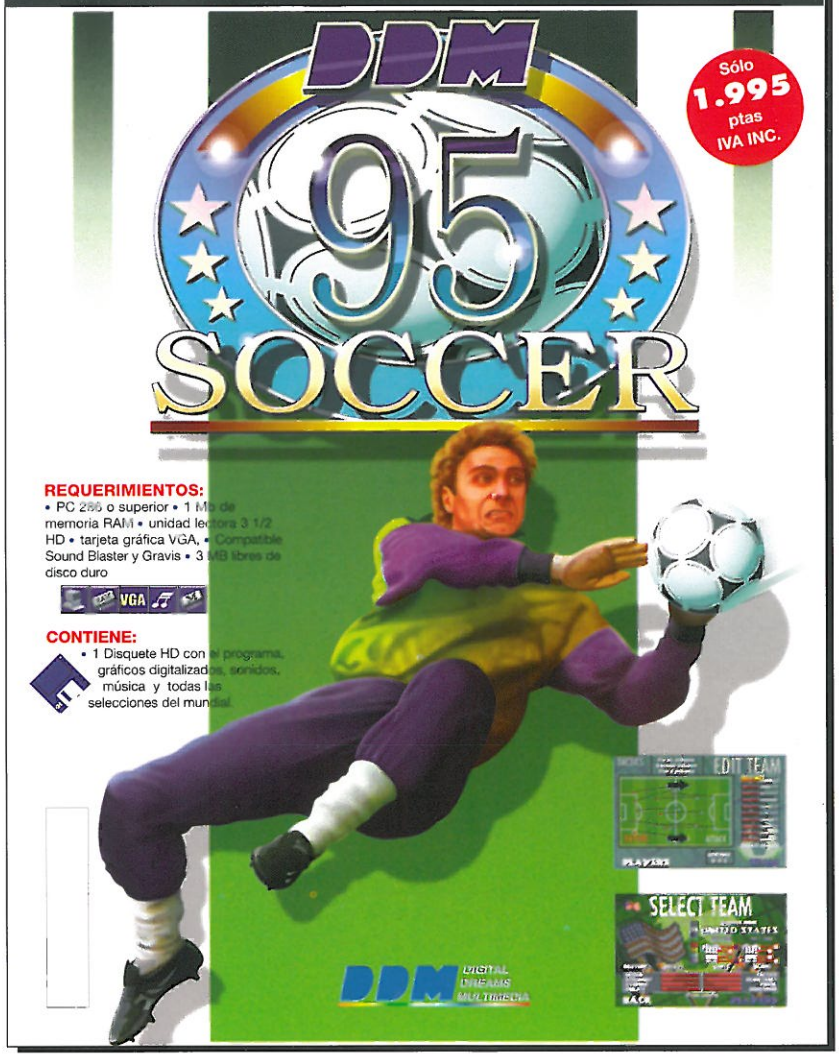
Todas las estrellas del fútbol en tú PC.

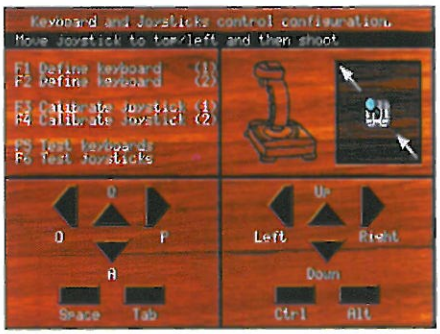
Un juego trepidante y divertido para todos.

Diseño de las tácticas individuales por jugador y equipos.

Apasionante sistema de juego de ligas.

PROGRAMA PARA PC. CONTIENE DISCO DE 3 1/2


DDM
95
SOCCER
Sólo 1.995 ptas IVA INC.
REQUERIMIENTOS:
• PC 286 o superior • 1 Mb de memoria RAM • unidad lectora 3 1/2 HD • tarjeta gráfica VGA, • Compatible Sound Blaster y Gravis • 3 MB libres de disco duro
VGA
CONTIENE:
• 1 Disquete HD con el programa, gráficos digitalizados, sonidos, música y todas las selecciones del mundial.
EDIT TEAM
SELECT TEAM
DDM DIGITAL DREAMS MULTIMEDIA



¡YA A LA VENTA EN TU QUIOSCO! POR SÓLO 1.995 PTAS

CON LA GARANTÍA DE DDM DIGITAL DREAMS MULTIMEDIA

ta DDM SOCCER 95 enviando este cupón o llamando al teléfono (91) 741.26.62 de 9 a 14 y de 15:30 a 18:30 o por Fax al (91) 3.20.60.72

que me envíen: ☐ **DDM SOCCER 95 por 1995 ptas, + 250 ptas. de gastos de envío.**

bre y apellidos.....Domicilio.....Población.....

incia.....C.P .....Fecha de nacimiento.....Profesión.....

MA DE PAGO:

Talón a ABETO EDITORIAL SL ☐ Contra-reembolso... Teléfono .....

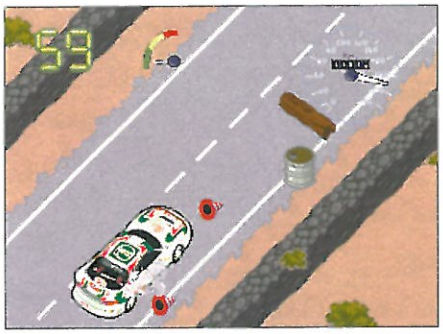
Giro Postal nº.....de fecha.....

Tarjeta de crédito VISA nº Firma ,

a de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

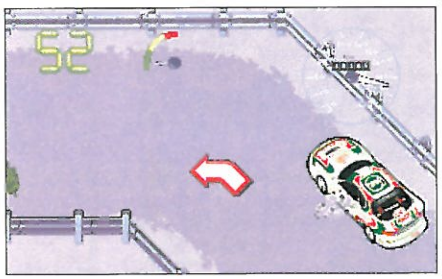
Rellena este cupón y envíalo a: ABETO EDITORIAL SL C/ Marques de Portugalete 10, Bajo 28027 Madrid.

TODA LA EMOCION DEL MUNDIAL DE RALLYS EN TU PC



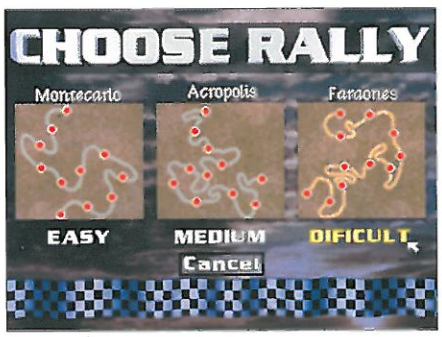
Evita los obstáculos con la ayuda de las instrucciones de tu copiloto.

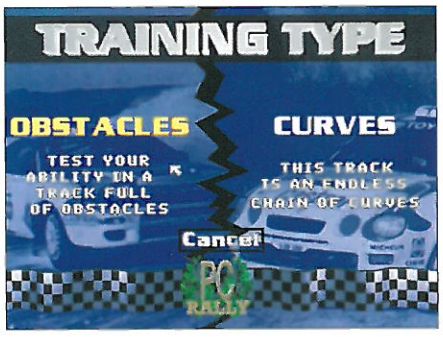
Gráficos y animaciones digitalizadas con sonido y música simultáneos.

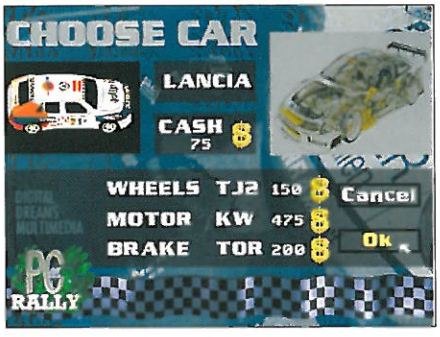


30 Tramos diferentes de nieve, tierra y asfalto para poner a prueba tus reflejos.

PC Rally
P.V.P. 2.495 ptas.
REQUERIMIENTOS:
• PC 386 o superior • 4 Mb de memoria RAM • unidad lectora 3 1/2 HD • tarjeta gráfica VGA, • Compatible Sound Blaster y Gravis • 8 MB libres de disco duro
CONTIENE:
• 2 Disquetes HD con el programa, gráficos digitalizados, 30 circuitos, efectos sonoros y música.
• Un completo manual que incluye datos sobre los pilotos, fotos, circuitos y calendario del mundial.
ROAD BOOK
30 Tramos diferentes con arena, nieve y asfalto para poner a prueba tus reflejos.
Conduce a toda velocidad el Toyota Célica o el Lancia Delta HF Integrale en los circuitos del mundial.
TOYOTA
Castrol
MICHELIN
CIBIE
TOWER COMMUNICATIONS
DDM DIGITAL DREAMS MULTIMEDIA







¡YA A LA VENTA EN TU QUIOSCO!
POR SÓLO 2495 PTAS

CON LA GARANTÍA DE DDM DIGITAL DREAMS MULTIMEDIA

Solicita PC RALLY enviando este cupón o llamando al teléfono (91) 741.26.62 de 9 a 14 y de 15:30 a 18:30 o por Fax al (91) 3.20.60.7

Deseo que me envíen: ☐ **PC RALLY por 2495 ptas, + 250 ptas. de gastos de envío.** ☐

Nombre y apellidos.....Domicilio.....Población.....

Provincia.....C.P .....Fecha de nacimiento.....Profesión.....

FORMA DE PAGO:

☐ Talón a TOWER COMMUNICATIONS SRL ☐ Contra-reembolso

☐ Giro Postal nº.....de fecha.....

Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

Teléfono .....

Firma ,

Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

TOWER COMMUNICATIONS

Rellena este cupón y envíalo a: TOWER COMMUNICATIONS SRL C/ Marqués de Portugalete, 10. Bajo